1997

Efficient processor management strategies for multicomputer systems

Chung-Yen Chang
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Efficient processor management strategies for multicomputer systems

by

Chung-Yen Chang

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Major Professor: Prasant Mohapatra

Iowa State University

Ames, Iowa

1997

Graduate College
Iowa State University

This is to certify that the Doctoral dissertation of

Chung-Yen Chang

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Signature was redacted for privacy.

For the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Prasant Mohapatra for his continuous support and guidance. I would also like to thank my committee members for their suggestions that improve the quality of this report. Special thanks go to Dr. Jim Davis for his kind helps when I needed them.

My family has always been supportive to me all these years. I appreciate their patience and understanding. I would like to thank my wife Fumin and daughter Tingyee for their love and tolerance. They make every day of my life enjoyable. I want to share the honor of earning the Ph.D. degree with my parents. Their encouragement and support make my pursuing of Ph.D. possible.

ABSTRACT

Multicomputers are cost-effective alternatives to the conventional supercomputers. Contemporary processor management schemes tend to underutilize the processors and leave many of the processors in the system idle while jobs are waiting for execution.

Instead of designing faster processors or interconnection networks, a substantial performance improvement can be obtained by implementing better processor management strategies. This dissertation studies the performance issues related to the processor management schemes and proposes several ways to enhance the multicomputer systems by means of processor management. The proposed schemes incorporate the concepts of size-reduction, non-contiguous allocation, as well as job migration. Job scheduling using a bypass-queue is also studied. All the proposed schemes are proven effective in improving the system performance via extensive simulations. Each proposed scheme has different implementation cost and constraints. In order to take advantage of these schemes, judicious selection of system parameters is important and is discussed.

1 OVERVIEW

Multicomputer systems provide cost-effective alternatives to the traditional supercomputers. A large number of processors are ensembled to build a multicomputer system. Complex tasks can be carried out efficiently through parallel operations on these processors. The performance of a multicomputer system depends on the cooperation among the processors. Efficient management of these processors is hence essential to exploit the potential of multicomputer systems.

1.1 Introduction

The basic building blocks of a multicomputer system is the processing unit. The number of processing unit in a multicomputer system ranges from tens to thousands or more. Each processing unit has its own execution units and memory. The processing units executes the instructions and obtains parallelism through the cooperation among multiple processor units. The cooperative effort among the processors is realized by passing messages among them.

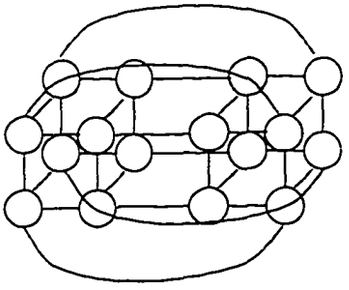
Processors are connected by an interconnection network in which messages travel from their sources to destinations. The interconnection network is either directly connected in a certain topology or implemented with a switch-based multistage interconnection network (MIN). Examples of multicomputer systems are illustrated in Figure 1.1. Every circle represents a processing unit which includes the execution unit and memory. Part (a) and (b) of Figure 1.1 are two of the most popular topologies, mesh and hyper-

cube. Example of commercial machines using these topologies including Cray T3D/T3E [1, 2], Intel Touchstone Delta [4], Paragon [3], and nCUBE [5]. Figure 1.1.(c) shows the organization of a MIN-based multicomputer. Examples of commercial machines using these organizations include IBM SP1/SP2 [6]. Among the topologies proposed for multicomputer architectures, the mesh topology has gained popularity because of its simplicity, regularity, and ease for VLSI implementation.

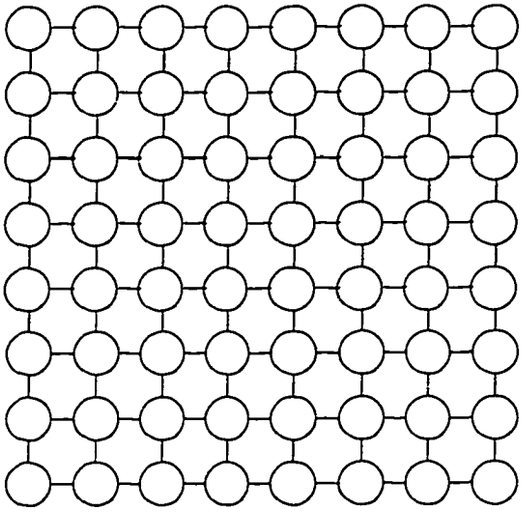
Processors in a multicomputer are shared among different processes. Jobs are allocated to a subset of processors for execution. Processor management concerns the allocation of the computational resources to jobs in the system. In a practical environment, multicomputers support a diverse mix of applications of various sizes and characteristics in a dynamic fashion. To support the diverse mix of applications, the processor management scheme implemented has to utilize the processing units as efficiently as possible. Two possible ways to achieve this goal include processor allocation and job scheduling.

Processor allocation involves selection of a set of processors for the execution of a job. Jobs of different characteristics require different number of processors in a predefined configuration (shape) similar to that of the system. The task of processor allocation has to fulfill these size and shape requirements. An executing job retains all the allocated nodes for the entire duration of its execution.

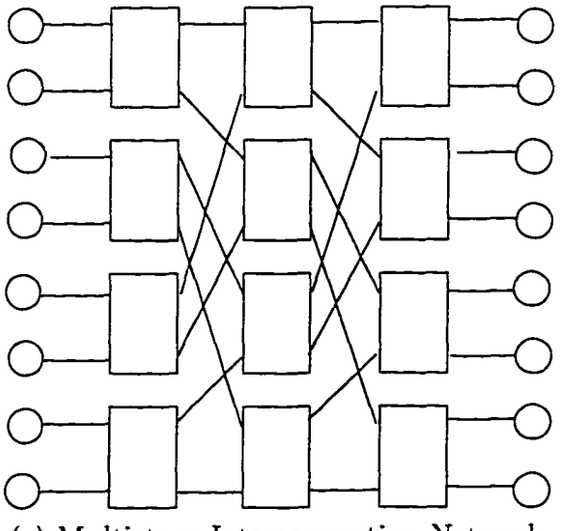
Several processor allocation algorithms have been proposed for the mesh-connected systems [7]–[26], and for hypercube-based systems [17]–[21] in the literature. Other researchers have proposed allocation algorithms [22]–[23] for the k-ary n-cube based system [24]. These algorithms allocate a job to a set of contiguous processing nodes to minimize the distance of communication paths and to avoid the interprocess interference. These allocation schemes are referred to as contiguous allocation schemes. Using these schemes, the allocator identifies the free nodes and decides whether the free nodes can form the required subgraph for the execution of a job. Allocation algorithms with better recognition ability for available subgraphs can improve the chance of assigning



(a) Hypercube



(b) Mesh



(c) Multistage Interconnection Network

Figure 1.1 Examples of Multicomputer System.

a job into the system and reduce the waiting delay. However, as studies [28, 29, 42] showed, significant performance improvement cannot be obtained by refining the conventional allocation algorithms. Because of the topological restriction, these algorithms suffer from fragmentation problem. Contiguous allocation leads to several fragmented groups of processors that cannot be used for the new tasks. Fragmentation is a serious problem associated with the allocation algorithms and is the main factor that limits their performance and will be discussed in detail in Section 2.1.4.

Non-contiguous allocation algorithms [25, 26] have been proposed to solve the fragmentation problem by allowing jobs to execute on non-contiguous nodes. By lifting the restriction on shape requirement, better utilization of the processing units is achieved. However, the strong correlation between the communication latency and processor allocation leaves the impact on job execution time difficult to estimate. Jobs allocated non-contiguously may incur high communication latency. Because of the unpredictable increase on communication latency caused by non-contiguous allocations, the non-contiguous allocation schemes may not be suitable for general application environments.

Job scheduling is another approach to improve the performance of a multicomputer system. It aims at reducing the queuing latency incurred by the jobs. The first-come-fist-serve (FCFS) scheduling is often chosen because it has low time complexity and is easy to implement. In the FCFS scheduling, a waiting job will block all the following jobs from being serviced even if there are idle processors in the system. It has been reported in [29, 30, 31, 32, 33, 34, 35] that by rearranging the sequence of jobs for execution, queuing delay can be reduced. A job scheduler chooses the next job for allocation and execution from the pending jobs. The blocking caused by fragmentation can therefore be reduced. Other researchers [37, 38] have considered implementing the concept of multiprogramming used in the uniprocessor environment for the multicomputer systems. These scheduling schemes have shown promising performance improvement. However,

they introduce significant overhead to the already complex allocation process. The high complexities of several allocation algorithms restrict the use of any complicated scheduling policy. In addition, the difference between the uniprocessor and multicomputer systems may cause the implementation of multiprogramming less desirable.

1.2 Goal

There are three areas of improvement that can be examined to enhance the performance of multicomputer systems. Faster processors speed up the execution of instructions and hence reduces the execution time. Faster interconnection network with higher bandwidth reduces the communication latency incurred by parallel jobs. Both methods require redesigning or upgrading of the hardware components and are costly. The third approach one can take is to implement better processor management strategy.

Studies have shown that the performance of multicomputer systems are limited by the inefficiency of the contemporary processor management policies. Mesh-connected systems are limited to a utilization of as low as 55% and hypercube systems are limited to about 70% utilization. In other words, the computational power of these systems are not fully exploited. Instead of pushing the technology on the processor design or pursuing a faster interconnection network, a considerable amount of improvement can be achieved by employing a better processor management policy so that the computational resources can be fully utilized.

The goal of this research is to improve the performance of multicomputer systems through the development of better processor management policies. The processor management approach, compared to others, is not only more cost-effective but also an essential way of utilizing the resources that are already built into the system. The schemes discussed in this dissertation are not only proposed from the performance perspective. They are also easy to implement and have low computational complexities. Many pre-

viously proposed schemes, although effective in producing promising performance, incur high implementation or computational complexities. The low complexity along with the high performance make the proposed processor management policies attractive.

1.3 Scope

The scope of this dissertation is focused on the processor management policies in multicomputer systems in a dynamic environment. We have chosen the mesh-connected system as our target architecture because of its wide availability and success in commercial market. Commercial systems such as the Intel Paragon, Touchstone Delta are constructed with the mesh topology. The research for processor management in mesh systems is also less mature compared to that of the hypercubes. However, the general concepts of these schemes are not limited to mesh systems. It is possible to modify the discussed schemes for other multicomputer systems.

Performance of a system can be evaluated through a variety of metrics. It could be latency of a job, utilization of a processing element, reliability of system, or throughput. In this dissertation, we consider performance from a users' point of view in which the average turnaround time is the main concern. Average turnaround time refers to the time between the submission of a job and the time when it terminates. It demonstrates the system's ability to handle the workload and reflects the services received by users (jobs).

We have analyzed the scheduling policy for a system in a dynamic environment. In a dynamic environment, the system has no knowledge about the execution time of a job until its termination. Scheduling for such a system is done with only the information about the number and shape of required processors. Static scheduling, in which the characteristics of jobs such as sizes and run-time are known prior to scheduling, or real-time scheduling, in which jobs have certain deadlines to meet, are complex topics by

themselves and are beyond the scope of this dissertation.

1.4 Approaches

The total time elapsed between the submission and completion of a job is called the turnaround time for the job. The turnaround time of a parallel application can be represented as a sum of three components,

$$T_{\text{total}} = T_{\text{queue}} + T_{\text{comp}} + T_{\text{comm}}. \tag{1.1}$$

The queuing delay, T_{queue}, represents the time a job spends in the queue. It is a function of the allocation algorithm and the scheduling strategy. The execution time of a job consists of computation time, T_{comp}, and communication latency, T_{comm}[1]. The computation time depends on the amount of computation required for an application. It also depends on the amount of parallelism and the number of processors allocated to a job. The communication latency is the time a job spent waiting for different subtasks to exchange information. It is a function of the communication pattern of application and the positions of allocated nodes.

By observing Equation 1.1 it is inferred that the key for better performance is to keep the three delay components as low as possible. T_{comp} depends on the job characteristics and the power of the processing elements. T_{comm} is affected by the communication pattern of the application and the design of the interconnection network. Improving the speed of the processing units helps reducing the computation time T_{comp}. With faster interconnection network, one can reduce the communication latency T_{comm}. These two improvements are not within the scope of processor management.

For a given multicomputer system, the speed of the processors and the interconnection network are fixed. Processor management policy does not reduce T_{comp} and T_{comm} directly. It is mainly targeted on reducing T_{queue}. However, the three delay

[1] T_{comm} is the part that is not already concurrent with T_{comp}.

components are correlated. It is possible that minimizing one delay component results in higher latency in one or the other two. For example, contiguous allocations focus on minimizing T_{comm}. Because of physical fragmentation associated with contiguous allocation schemes, the system is underutilized and a large T_{queue} is observed. Non-contiguous allocation schemes reduce T_{queue} by eliminating the fragmentation problem. However, jobs may incur higher T_{comm} in these schemes because of the longer communication path and interprocess interference caused by allocating processors non-contiguously. It is also possible to multiprogram jobs on a set of processor so that queuing latency is reduced. Doing so causes the computation time to increase because the computational power of the processors is shared among multiple jobs.

The queuing delay dominates the other two delay components in the turnaround time of a job in a medium to heavily-loaded system. The average turnaround time of jobs can be reduced if T_{queue} can be reduced. A good processor management policy reduces the queuing delay without introducing much overhead in the computation time and communication latency.

This dissertation proposes several efficient processor management strategies for multicomputer systems based on the trade-offs among the three delay components. Two novel processor allocation schemes are proposed. A *restricted size reduction (RSR)* scheme is developed based on the trade-off between the computation time and the queuing delay. The job size reduction technique incurs higher job execution time because of the number of processors assigned to the execution of size-reduced jobs are less than the number requested. An *adaptive non-contiguous allocation (ANCA)* algorithm allocates jobs non-contiguously when fragmentation prevents the job from being allocated. The non-contiguous allocation scheme violates the optimal communication paths and patterns in the original requests and may cause higher communication latencies. These two processor allocation schemes are successfully in reducing queuing latency while keeping the increase on the other delay components minimal.

Rearranging the sequence of jobs for execution is able to reduce the queuing delay without introducing overhead in the computation time or communication latency. A scheduling policy based on a *bypass-queue (BQ)* technique is studied. It is effective in reducing queuing delay but may causes the increase on number of allocation attempts for the same stream of incoming jobs. To overcome this complexity problem, a *fixed-orientation (FO)* allocation scheme which has low complexity yet efficient, is proposed to be combined with the BQ scheme.

Another way to improve the performance via processor management is by performing job migrations in order to reduce the fragmentation problem. Three variations of a job migration scheme is proposed and compared. Performing job migration at the completion of a job produces the lowest average turnaround time compared to migrating jobs at an allocation failure or migrating at both completion and allocation failure.

A hybrid allocation scheme which combines the conventional allocation with the two novel approaches, *RSR* and *ANCA* is also included in this report. The idea of a hybrid allocation is to handle the processor management in an environment where some of the jobs are sensitive to size-reduction or non-contiguous allocation. With the assistance of user directives, the penalty associated with the two novel allocation approaches is expected to be minimized and the system as a whole can benefit from the effective use of the three allocation schemes. Interesting results are observed in this study and a modified RSR algorithm is proposed to be used in such an environment.

1.5 Organization of the Dissertation

This dissertation is organized as follows. The next chapter surveys the related works in the field of processor management in multicomputer systems. Comparison of the contemporary schemes are discussed and their common problem is analyzed. Based on the understanding of the problems associated with the contemporary processor management

schemes, several new processor management strategies are proposed throughout the rest of this work.

Chapter 3 presents the reduced size reduction allocation scheme. Chapter 4 is dedicated to the study on the effect of processor management to the communication latency in multicomputer systems followed by the adaptive non-contiguous allocation in Chapter 5. Chapter 6 studies the integration between processor allocation and job scheduling using the fixed-orientation allocation and bypass-queue scheduling. Chapter 7 evaluates different approaches of job migration to reduce the blocking delay caused by the fragmentation. The last processor management scheme proposed in this work in Chapter 8 considers an environment in which some of the jobs are not suitable for size-reduction or non-contiguous allocation and therefore user directives have to be incorporated. Performance evaluations of the proposed schemes are carried out by simulation and are reported in each chapter.

2 RELATED WORKS

Many researchers have attempted to improve the performance of processor management schemes for multicomputers. Two main tracks have been taken independently; improving allocation algorithms and developing efficient scheduling policies. A few other approaches such as job migration has also been proposed. This chapter surveys these approaches, their characteristics, their applicabilities, and their inadequacies.

2.1 Processor Allocation

Processor allocation schemes have been extensively studied by many researchers. This dissertation is focused mainly on the mesh systems, only processor allocation schemes for the mesh systems are surveyed in details. Processor allocation schemes for hypercubes and other topologies are included with less details. The allocation schemes are classified as either contiguous or non-contiguous. Some allocation schemes use neither of these strategies and we have classified them into the other allocation approaches.

2.1.1 Allocation Schemes for Mesh Systems

Contiguous allocation algorithms for mesh systems search for available submeshes in the system to execute the jobs. They are limited by the size and shape requirement of incoming jobs. Non-contiguous allocation schemes loose the shape requirement to achieve higher utilization of the system. We outline these allocation schemes proposed in the literature.

2.1.1.1 Contiguous Allocation for Mesh

The rationale behind the contiguous allocation schemes is to minimize the communication latency incurred by jobs. By allocating a job to a contiguous set of processors, the distance of the communication path and the amount of interprocess interference is minimized. In addition, parallel programs are often optimized for the machine architecture. By maintaining the allocated nodes in a shape that is requested, the communication patterns in a job is most likely to be optimized.

Two Dimensional Buddy: The two dimensional buddy *(TDB)* [7] is a generalization of the one dimensional buddy algorithm [17] proposed for storage allocation. The system is assumed to be a square whose side lengths equal to a power of two. Jobs are assumed to request square submeshes. The size of a requested submesh is rounded up to the nearest power of two and are allocated a submesh of the corresponding size. Every square submesh can form a larger square submesh with its three neighboring buddies. Jobs are allocated to the buddies of processors. The TDB allocation suffers from internal fragmentation because it only allocates square submeshes with the side lengths equal to a power of 2. It is also not applicable for general systems such as the Intel Touchstone Delta and the Intel Paragon because of its system shape requirement (the configuration of Touchstone Delta and Paragon are not necessarily square). The time complexity for allocating an $(N \times N)$ job in an $(M \times M)$ mesh when there are k jobs in the system is equal to $O(k(\frac{M}{N})^2)$.

Frame Sliding: The frame sliding method [8] is proposed to reduce the fragmentation problem of the TDB allocation. It can be used for meshes of any size. The requested submesh size and shape for a job is considered as a frame. A frame filled with free nodes is considered available for the execution of the job. The algorithm slides the frame across the system at non-overlapping locations to examine for a free submesh for the allocation of the job. The checking can be done by verifying a bit-array which represents the status

of the processors. The complexity of this allocation for an $(m \times n)$ job in an $(M \times N)$ system is equal to $O(\frac{MN}{mn})$.

First-fit and Best-fit: The *first-fit* and *best-fit* algorithms [9] guarantee that a required submesh can always be identified provided it exists. However, they fail to locate the available submesh if the available submesh has different orientation than the required one. These two algorithms scan a bit-array for the allocation process. Both algorithm perform equally well at the same time complexity of $O(MN)$ in a $(M \times N)$ system with the space complexity of $O(MN)$.

Adaptive-Scan: To further enhance the allocation ability, the adaptive-scan [10] changes the orientation of a submesh when the required size and shape of submesh is not available. By rotating the submesh request, the possibility of allocating a job is increased and the performance is thus improved. This algorithm reduces the number of steps in scanning the system for available processors by skipping the checking of unallocable nodes.

Busy-List: A busy-list allocation scheme [11] is different from the other allocation algorithms in the sense that it uses a busy-list for the checking process instead of the bit-array used in other algorithms. It has the same submesh recognition ability as that of the adaptive-scan policy.

Free-List: Free-list algorithm utilizing a list for processor allocation [12]. The free-list scheme maintains a list of free submeshes for the allocation. Its allocation process has a lower time complexity than that of the busy-list scheme but its deallocation time complexity is higher. Its submesh recognition ability is similar to that of the adaptive-scan and busy-list algorithms.

Quick Allocation: An algorithm called *Quick Allocation* is recently proposed in [13] to allocate processors in a two-dimensional mesh environment. The basic idea of the quick allocation is similar to the first-fit scheme except the overhead for searching the available submesh is reduced. A set of adjacent processors called *covered segment* is identified for

each row. Each covered segment indicates if the corresponding row has a potential base for the allocation. Using the covered segment, the checking complexity is significantly reduced and the allocation time is proven via simulation to be the lowest.

Stack Allocation: An allocation algorithm based on a busy list of the allocated submesh and a stack-oriented search is proposed in [15]. The reject set and cover set corresponding to the job to be allocated is constructed using the busy list. The reject set and cover set represent the locations where the base for the new submesh cannot be allocated. A process called spatial subtraction is then used to subtract the reject and cover sets from the possible locations for the base of the new job. The search process is speeded up by using a stack and the allocation time is shown to be even shorter than the quick allocation.

2.1.1.2 Non-contiguous Allocation Algorithms for Mesh

Hardware advances such as wormhole routing and faster switching techniques have made the communication latency less sensitive to the distance between the communicating nodes. Several non-contiguous allocation algorithms have been proposed for the mesh in [25] to utilize this concept. These include the *naive, random, paging,* and *multiple buddy system (MBS)* The naive and random strategies allocate jobs based solely on the size requirement. Random allocation is a straightforward strategy in which a job requesting p processors is satisfied with p randomly selected nodes. Naive algorithm allocates a request for p processors to the first p free nodes found in a row major scan. A job is never denied service if there is enough processors in the system regardless of the contiguity of the nodes.

The paging allocation maintains a partial contiguity of the allocated processors by allocating predefined pages to a job. Internal fragmentation may occur if the number of processors requested by a job is not a multiple of a page. The multiple buddy strategy is an extension of the 2-D buddy strategy. The system is divided into non-overlapped

square submeshes with side-lengths equal to powers of 2 upon initialization. The number of processors, p, requested by an incoming job is factorized into a base 4 representation of $\sum_{i=0}^{\lfloor \log_4 p \rfloor} d_i \times (2^i \times 2^i)$, where $0 \leq d_i \leq 3$. The request is then allocated to the system according to the factorized number in which d_i number of $2^i \times 2^i$ blocks are required. If a required block is unavailable, MBS recursively searches for a bigger block and repeatedly breaks it down into buddies until it produces blocks of the desired size. If that fails, the requested block is broken into 4 requests for smaller blocks and the searching process repeats.

2.1.2 Allocation Schemes for Hypercube

2.1.2.1 Contiguous Allocation for Hypercube

Buddy Allocation: Buddy allocation [17] originally proposed for memory allocation is implemented on the nCUBE system for processor allocation. Every subcube has a buddy of the same size. Two adjacent buddies can be combined to form a larger cube for the execution of a larger job. A job is always assigned a subcube for its execution. For an n-cube, the nodes are numbered from 0 to $2^n - 1$. For a job requiring a k cube, the algorithm searches for the least integer m such that all the nodes in the region $[m2^k, (m+1)2^k - 1]$ are available. This scheme does not provide a complete subcube recognition ability in a dynamic environment. The time complexity of allocation and deallocation in the above case are $O(2^n)$ and $O(2^k)$, respectively. By using an efficient data structure [39], the complexity of allocation and deallocation can be reduced to $O(n)$.

Other Allocation Algorithms for Hypercube: Many other allocation algorithms have been proposed for the hypercubes to implement perfect subcube recognition ability. All these algorithms allocate jobs based on the free subcubes as does in the buddy allocation. Some examples include the multiple gray code [18], free list [19], tree collapsing

[20] and PC-graph [21]. These algorithms implement high subcube recognition abilities at the price of implementation complexities. The detailed discussion of these algorithms can be found in the literature [18]–[21].

2.1.2.2 Non-contiguous Allocation for Hypercube

Three processor allocation algorithms are proposed for the k-ary n-cube in [22], namely the k-ary partner, non-contiguous Multiple Buddy and Multiple Partner strategies. The k-ary Partner strategy is a conventional contiguous processor allocation strategy that improves subcube recognition. The non-contiguous Multiple Buddy and Multiple Partner strategies address the problem of fragmentation by allocating jobs to non-contiguous processors. These algorithms are applicable to the hypercube systems when k is set to 2. Another non-contiguous allocation algorithm based on the busy-list approach is proposed in [27]

2.1.3 Other Allocation Approaches

A few allocation approaches are not clearly identified by the classification of contiguous or non-contiguous allocation. A rather unconventional approach taken to improve the performance of the multicomputer system is by changing the size of the jobs. Changing the job size to avoid fragmentation and reduce the queuing delay has been studied in [40, 41, 42]. In [41], the authors proposed two allocation policies for the mesh-connected system, namely the *equi-partition* and *folding* allocations. Both policies assumes the initial submesh requirements are all equal to the size of the system. Jobs are also migrated between nodes in the system. The job size assumption is not practical and the overhead for migrating jobs are not ignorable. The two methods in [41] are hence less attractive for the actual implementation.

The *limit* allocation is proposed for the hypercubes in [42] and is claimed to be the most efficient processor management strategy for the hypercube systems. Three limit

allocation algorithms are proposed, namely the *limit-k, greedy,* and *average.* The basic idea of limit allocation is to reduce fragmentation by limiting the maximum job size in the system. In all but the greedy limit scheme, jobs can only be executed on subcubes smaller than a limited size. Job that requires a subcube larger than the limit will have to be folded to the limited size. This causes underutilization of the system under low load and results in poor performance. The major problem with the limit allocation is its unfair treatment to jobs of different sizes. A folded job is executed on less processors then it initially requested. This increases the load of every processor and the execution time of the job is expected to increase. Comparing with the execution time of the unfolded jobs, the folded jobs are obviously treated unfairly. The *limit* - Δ k allocation [38] is an extension of the limit allocation to the mesh system. It searches for smaller submeshes when the required submesh is not present in the system.

2.1.4 Fragmentation Problem

The main problem associated with the allocation algorithms is the fragmentation problem which prevents the idle processors from being utilized. Fragmentation can be classified as internal and external. The *internal fragmentation* is a result of allocating jobs only to certain size submeshes. When a job is assigned to more processors than it requires, the extra nodes allocated are not used for actual computation and are wasted. This happens when a job requests a submesh that does not fit the requirement of the allocation algorithm. As an example, internal fragmentation occurs when a job does not require a square submesh with sides equal to power of two and is allocated using the TDB scheme.

External fragmentation occurs when the allocation scheme cannot allocate the available processors to the incoming jobs. It can be further divided into three types based on their causes. The first type of external fragmentation is called *insufficient resource fragmentation.* Every job requires a certain number of processors for its execution. If

the available processors in the system is less than the number required, the job cannot be allocated. The second type of external fragmentation is a result of imperfect recognition ability of the allocation algorithms. A suitable submesh may exist for execution of a job while the allocation algorithm fails to locate the available submesh. This type of external fragmentation can be found in the TDB and the frame-sliding algorithms (if the frame slides through more than one hop at a time). We refer to this type of external fragmentation as *virtual fragmentation.* The third type of external fragmentation is caused by dynamic departures of jobs. Nodes released by the terminated jobs can be scattered in the mesh. They may not form a submesh big enough to accommodate the incoming task although the number of free nodes may be sufficient. This type of fragmentation is therefore referred to as *physical fragmentation.* Figure 2.1 shows a mesh system with three busy submeshes highlighted in different shades. There are 27 free nodes in this mesh. Using the conventional schemes, any job that requires more than 27 nodes cannot be allocated because of insufficient resource fragmentation. Suppose a job requires a 2x3 submesh (2 rows, 3 columns). Using the frame-sliding algorithm (where the frame slides by the height and width of the required submesh size), it will miss the free 2x3 submesh in the mesh because of its imperfect recognition ability. This is an example of virtual fragmentation. An example of physical fragmentation can be observed from the same figure when a 4x5 submesh is required. The 27 available nodes do not form the required shape and therefore will not be allocated with conventional allocation schemes.

The allocation schemes and their associated fragmentation are listed in Table 2.1. The performance of an allocation scheme is inversely proportional to the fragmentation it creates in the system. TDB is the only scheme listed with all fragmentation. Its performance is also the worst compared to the other allocation schemes in a dynamic system environment. Frame-sliding has physical and may have virtual fragmentation. The first-fit and best-fit algorithms have virtual fragmentation only when the submesh request and the available submesh have different orientations. Their performance is slightly

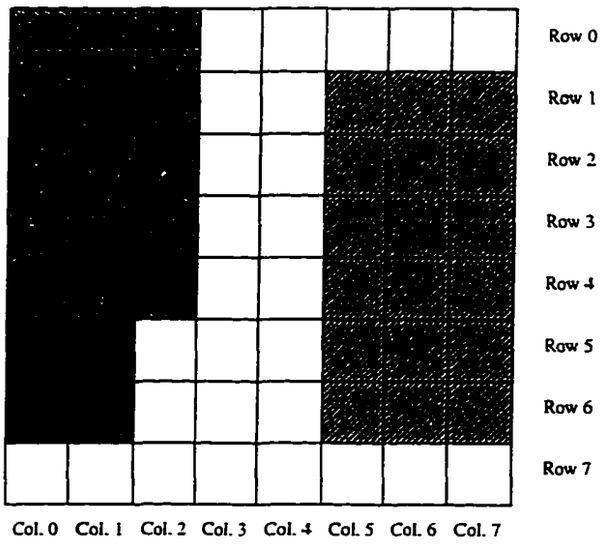


Figure 2.1 Fragmentation Problems.

worse than the other allocations that do not have virtual fragmentation. However, all conventional algorithms cannot deal with physical fragmentation. RSR, non-contiguous allocation, and ANCA are the only schemes that can reduce the physical fragmentation.

2.2 Scheduling Approaches

A variety of scheduling schemes for the multicomputers have been reported in the literature [29]–[38] and [43]–[52]. Traditionally, scheduling in multicomputers is done via two approaches, multiprogramming or rearranging the sequence of jobs. Some previous works have extended the scheduling problems in multicomputer systems to include the real-time jobs.

2.2.1 Rearranging the Sequence of Execution

Research related to job scheduling in multicomputers has been focused on arranging the order of execution of the waiting jobs. Because of the blockade situation caused by FCFS discipline, the processors are usually underutilized. Several processors can be

Table 2.1 Fragmentation of Various Allocation Schemes.

Scheme	Internal Fragmentation	Insufficient Resource	Virtual Fragmentation	Physical Fragmentation
TDB	Yes	Yes	Yes	Yes
Frame-Sliding	No	Yes	Yes	Yes
FF and BF	No	Yes	Some[a]	Yes
Adaptive-Scan	No	Yes	No	Yes
Busy-List	No	Yes	No	Yes
Free-List	No	Yes	No	Yes
RSR	No	Some[b]	No	Some[c]
Non-Contiguous	No	Yes	No	No
ANCA	No	Yes	No	Some[d]

[a]Only happens when the available submesh is in different orientation.
[b]Can be completely removed if size reduction allows all jobs to be executed on a single node.
[c]Can be completely removed if size reduction allows all jobs to be executed on a single node.
[d]Can be completely removed with adaptability set to allow all jobs to be allocated totally non-contiguously.

left idle even when there are jobs waiting for execution. By executing the jobs in a carefully arranged order, the blocking effect can be reduced. Examples of scheduling policies taking this approach include the schemes reported in [29]–[35].

The *scan* scheduling policy [29] is a successful example of job scheduling scheme for the hypercube system. It allocates jobs of the same dimension together to avoid the fragmentation of the system. A similar reservation scheduling is proposed for the hypercubes in [31]. It attempts to improve the performance by allowing jobs to bypass the waiting ones. To preserve fairness, a reservation mechanism is used to ensure the waiting processes get the requested services after the completion of jobs running on their reserved subcubes. The *lazy* [32] scheduling temporarily delays the allocation of a job if any other job of the same dimension is running in a hypercube. The delayed job is then executed on the existing subcube rather than acquiring a new subcube. The fragmentation of the system and the blocking problem with the FCFS scheme are both reduced.

In [33], a scheduling policy incorporating the *priority* and *reservation* techniques is

proposed. The priority scheme is a variant of the priority scheduling in the uniprocessor environment. The reservation method allows an unallocable job to reserve a submesh in the system and allow other jobs to bypass it to avoid the blocking situation. The reservation scheme may cause underutilization of the system when the reserved submesh has different size than the executing submeshes. A *between-within queue (BWQ)* scheme is proposed in [34]. The BWQ scheduling imitates the idea of *scan* scheduling [29] in the hypercube by segregating jobs according to their shapes and sizes. Jobs of similar sizes and shapes wait in the same queue for allocation and the system keeps several such queues for different sizes and shapes. The *HELM* scheduling [35] arranges jobs in three different queues, namely the *Entrance queue*, *Lookahead queue* and *High Priority queue*. Jobs are dispatched from these three queues according to different service disciplines.

2.2.2 Multiprogramming in Multicomputers

The concept of multiprogramming has been implemented in the M^2 [38] and the *TSS* [37] strategies. Both policies require large memory space to store the information of multiprogrammed nodes and incur high time complexities. In addition, there are several disadvantages for using the multiprogramming concept along with space sharing (through partitioning) in a multicomputer. First, the memory space on each processor is usually smaller than that attached to a single processor system thus the number of multiprogrammable jobs is limited. Second, jobs come in different shapes and sizes in a dynamic system. Multiprogramming jobs of different sizes and shapes causes some processors to be underutilized when the system context switches to a different job. Third, the synchronization of context switching is difficult to implement with the large number of processors involved. Poorly synchronized system can result in unnecessary waiting or even deadlock configurations. In addition, when jobs are swapped in and out from the disks, excessive traffic is introduced into the interconnection network. This is a tremendous overhead and cannot be ignored. The multiprogramming policies (M^2 and TSS)

fail to address the above issues and thus may not be practical for real implementation.

These scheduling policies also require complicated operations to be performed in addition to the allocation process. These operations introduce significant overhead in the system. Another overhead caused by most of the scheduling policies is the increased number of allocation attempts. Because the scheduling strategies try to utilize more processors in the system, the scheduler causes the allocator to check the allocation of more jobs to utilize the available processors. The increased number of allocation attempts incurs very high overhead because of the high computation complexity of the allocation algorithm.

2.2.3 Problems of Contemporary Scheduling Schemes

There are several problems associated with these scheduling approaches. A common problem associated with all the scheduling strategies discussed is that they require excessive storage for implementation. The multiple queues and the special data structure required to implement these schemes pose a storage problem. The schedulers using the reservation scheme requires additional queues to keep track of the jobs holding a reservation. For the lazy scheduling, separate queues are required for different size jobs. This is also true for the scan policy. Second, the complexities of the scheduling approaches are high. In addition to the underlying allocation algorithm used, the scheduler imposes additional overhead for determining the order of the execution. Other than the high complexity and extra storage requirement, the scheduling approaches fail to guarantee a promising performance. In the reservation approach, a node can be idle waiting for the availability of other nodes that are reserved together for the execution of the same job. This causes the system to be underutilized and limits the performance. The performance gain of the scan policy is dependent on the workload environment. For example, it does not provide a significant performance gain when the service time distribution is hyper-exponential. There are also problems associated with the multiprogramming

approaches as discussed in Section 2.2.2.

2.2.4 Real-Time Scheduling Schemes

Several scheduling schemes [43]–[52] are proposed for scheduling real-time jobs in multicomputers. The difference between these scheduling schemes and the ones discussed in this dissertation is on the time constraint. In real-time systems, jobs have deadlines to make. The criteria for a good real-time scheduling strategy is to minimize the fraction of jobs missing their deadlines. The execution time and the deadline requirement of a job is known before entering the system so that the scheduler can perform the scheduling based on the size and deadline requirements. The scheduling policies in this work deal with the dynamic system in which jobs come into the system dynamically. The only known properties about a job at submission is its size and shape requirement and the optimization criteria for the scheduling is to minimize the average turnaround time of jobs in the system.

2.3 Other Processor Management Approaches

Job migration has been proposed in [53] to solve the fragmentation for the hypercubes. By moving jobs to one side of the system, the fragmentation is expected to be reduced. The problem associated with the job migration approach is the selection of migration path. A path across another process may interfere with the operation of that process. Therefore the migration path has to be properly selected.

Migrating jobs causes other concerns. For instance, the rebuilt of the working set in the cache is an overhead that has to be considered. In addition, once migrated, the synchronization among all processors allocated to a job may be lost. Checkpointing mechanisms have to be enforced to maintain proper synchronization among the allocated processors.

The schemes discussed in [41] also facilitate the idea of moving the processes in the system. In addition, the equi-partition and folding approach also suggest to dynamically change the number of processors allocated to a job. These schemes not only face the problems associated with migrating jobs, also cause the concern on the feasibility of implementation. The available parallelism in a job may not allow the dynamic changing of allocated processor numbers.

3 RESTRICTED SIZE REDUCTION (RSR) SCHEME

3.1 Introduction

To reduce the effect of fragmentation, the RSR scheme adaptively allocates jobs to smaller size submeshes when fragmentation prevents them from being allocated. The tradeoff for this scheme is between the queuing delay and the longer execution time caused by executing jobs on smaller number of processors. The number of times that size reduction can be applied to a job is restricted to minimize the side-effect of the increased execution time caused by the size reduction. The allocation method is thus called the *restricted size reduction (RSR)* method.

Size-reduction is a straightforward process in which a submesh is folded along its longer side. If a hypercube system is considered, the folding is done by allocating the job to a smaller dimension subcube which has only half of the processors. We use an example to show how the performance of a multicomputer can be improved by reducing the job size. Fig. 3.1 shows a 3-cube system in which the nodes $(0,1)$ and $(6,7)$ are executing two different processes. Assume that a job which requires a 2-cube for its execution is submitted to the system. Although there are sufficient processors to form a 2-cube, no allocation algorithm can allocate this 2-cube job into the system because the four available processors are fragmented in two disjoint 1-cubes. The requesting job hence has to wait until either subcube $(0,1)$ or $(6,7)$ are released. On the other hand, we can fold the 2-cube job into a 1-cube job which requires only two nodes. Then the requesting job can start its execution immediately on nodes $(2,3)$ or $(4,5)$ and the

fragmented nodes are utilized. The fragmentation in other topologies can also be reduced in a similar fashion. Executing a job on a small number of nodes adaptively benefits the performance in two ways. First, physically fragmented nodes are utilized which, in turn, helps in accommodating more number of jobs in the system. Physical and insufficient resource fragmentation are thus reduced. Second, waiting delay is reduced because jobs can be allocated earlier. Quite often it might be advantageous to execute a task paying a penalty of execution time than to wait for the availability of the required size and shape of submesh.

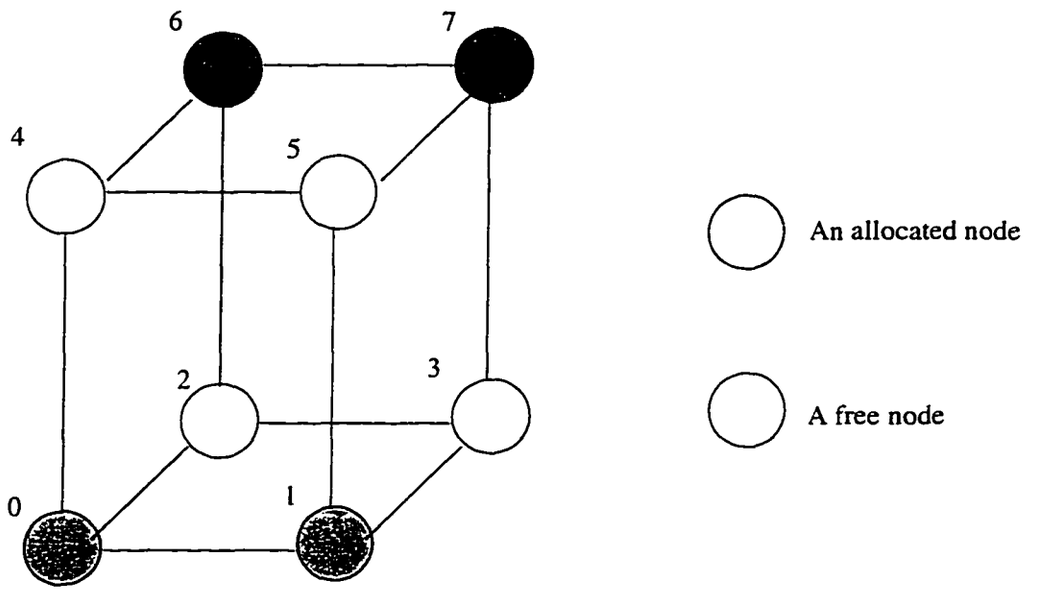


Figure 3.1 Reducing Fragmentation in a Hypercube by Job Size Reduction.

The RSR scheme exploits this observation and has the following important characteristics.

1. The RSR scheme is a generic processor management concept and is not limited to a single architecture or a particular allocation algorithm.

2. It is adaptive. A job is only folded when fragmentation prevents it from execution. A job is always assigned the system resources it requested when the resources

are available. This property guarantees that the system maintains a reasonable utilization at all ranges of workload.

3. It is flexible. System administrators can determine the optimal restriction on the size reduction according to the individual system's need and workload. Individual job can also specify the number of size reduction it can tolerate to avoid performance degradation.

4. It has low storage and computation complexity compared to other approaches.

3.2 Suitability of Size-Reduction

Suitability of size-reduction needs to be addressed for the practicability of the RSR scheme. In a multicomputer system, jobs come in different sizes according to their inherent parallelism and resources requests. The inherent parallelism of a job prohibits changing the job size randomly. However, we argue that it is safe to scale down a job in a regular fashion. Degree of parallelism limits the maximum number of subtasks that can be run in parallel. It is always possible to reduce the number of concurrently running subtasks than to increase it. The nCUBE's software environment [65] explicitly supports execution of a job on different size cubes. A program can be executed on a subcube larger or smaller than the one specified when it was compiled. For applications that require at least some certain number of processors to execute, it is still possible to fold the program and run them on a smaller subsystem in a context-switching fashion. However, the amount of size reduction is always restricted by the memory requirement of the job and the memories available at the nodes. The RSR algorithms never reduces the size requirement of a job if the available memory is insufficient.

Executing jobs on a reduced size submesh causes execution time to be longer. General speedup studies [67, 68] state that for most of the applications parallel computers provide

sub-linear speedups. Therefore, when a job is folded to half of the size it requested, it is unlikely for the job's execution time to exceed twice of its execution time when its request was granted without folding. Additionally, the communication overhead for less processors is expected to be reduced for several reasons. First, the communication path is shorter in a smaller subsystem. Second, the smaller number of processors in the smaller subsystem causes less interference between messages transmitted by different processors. Third, the frequency of interprocessor communication could be reduced because each processor now executes a larger share of information. In the case of multiple copies of program from the same task running on the same processor by context-switching, the communication overhead could be even less because some inter-processor communication might become intra-processor communication. Conservatively, it is safe to assume a linear increase on execution time when a job is folded as is assumed in [42].

3.3 The RSR Algorithm

RSR scheme consists of two components, an underlying allocation algorithm and a restriction on number of size reductions that can be applied to a job. RSR scheme allowing at most t times of size reduction of a job is referred to as *RSR-t* allocation. A job that gets to the head of the job queue is examined for allocation. If the embedded allocation algorithm finds a suitable subsystem of free processors for the execution of the job, the job is allocated for execution. If a subsystem of the required size cannot be located, the allocator reduces the size of the job to the next smaller allocable subsystem and examine the availability of free nodes for the job's execution. This process repeats for a job until either the job is allocated or the number of times of size reduction *(t)* of the job is reached. The queue is only stopped when the job at the head of the queue cannot be allocated into the system after all allowable size reductions have been considered. Once a job releases the nodes it holds, the allocation process will repeat until all the

jobs in queue is allocated or the queue is blocked.

The RSR scheme is very flexible in the sense that any architecture and allocation algorithms can benefit from this scheme. The reason for the restriction on job size reduction is to ensure the performance gain of reducing fragmentation is not outweighed by the loss of reduced parallelism in the application. The system administrator can determine the maximum number of folding that a job can endure based on the system status and the workload parameters to get the optimal performance. Individual jobs can also specify their own restrictions to avoid unnecessary increase on the execution time. A job is only folded when fragmentation prevents it from execution. Jobs of all sizes have the possibility of being folded. Fragmentation is greatly reduced using RSR. The internal fragmentation and virtual fragmentation is associated with the underlying algorithm. If the chosen algorithm is free from internal and virtual fragmentation, there will be no internal and virtual fragmentation using RSR. Physical fragmentation is reduced because the fragmented nodes can be utilized with reduced size jobs. In the extreme case, when all jobs are allowed to be executed on a single processor, there will be no fragmentation of any kind.

To describe the RSR algorithm for a particular architecture, two things have to be considered, the embedded allocation algorithm and the restriction of the size reduction. Any existing allocation algorithm can be used with the RSR scheme. Our results show that a simple algorithm with the RSR technique can easily outperform the more robust allocation algorithms. Therefore, it is advantageous to choose an algorithm with lower complexity. The size reduction is done depending on the embedded allocation algorithm. For instance in the mesh systems with TDB algorithm, reducing the size of a job once results in a smaller square which requires 1/4 of the processors it requested before the reduction. For all allocation algorithms in the hypercubes, a size reduction folds a job into a smaller cube with half the number of the processors. The restriction of the number of times that size reduction can be applied to a job is a parameter of system load and

performance. An RSR allocation with the maximum times of size reduction allowed to a job set to t is called *RSR-t allocation*. The size of a job is guaranteed to be reduced less than t times to ensure that the performance gain. The sketch of the RSR-t allocation in the hypercube system is shown below.

Step 1 Let k be the size of the subcube requested by the job to be allocated. Set the minimum allowable size $s = MIN\{k - t, 0\}$.

Step 2 Check the availability of the k-cube using the embedded allocation algorithm. If found, allocate the job and goto step 4.

Step 3 Set k to $k - 1$. If $k >= s$ goto step 2, else goto step 5.

Step 4 If the job queue is not empty, goto step 1 to allocate the first job in the queue.

Step 5 End

Figure 3.2 The RSR-t Allocation for Hypercube

Judicious selection of the value of t is essential to exploit the advantages offered by the RSR allocation scheme. Some pointers toward the selection of the value of t are discussed in Section 3.4.2.1. Reduction of job size beyond a certain times provides negligible performance improvement (if at all) with the increase in complexity. In the hypercube system, if the maximum number of folding allowed is set such that all the jobs can be assigned to a single node when necessary, the RSR allocation is reduces to the greedy allocation scheme reported in [42].

3.3.1 Complexity Analysis

Complexity of the RSR allocation depends on the underlying allocation algorithm used. In a *RSR-t* allocation, the underlying allocation algorithm is called at most t times to check the availability of free processors. The worst case for the complexity of the RSR-t allocation algorithm would be $O(t \times C(x))$, where $C(x)$ is the complexity

of the underlying algorithm. As observed later, a maximum of 2 to 4 size reduction is sufficient; thus adding very little to the complexity of the underlying allocation schemes. In many cases, the checking for all the different sizes can be done in one pass. For example, if the buddy allocation is used for the RSR-t allocation in hypercube, instead of checking the available subcubes with sizes in descending order, one can check the subcube in ascending order from the minimum allowable size. A set of t variables can be used as temporary storage for the possible locations of subcubes of different sizes. The allocator can then assign the job according to the information stored in these temporary variables. If the checking of all different size subsystems can be done in one pass as stated above, the complexity of the RSR allocation is equal to the complexity of the underlying allocation algorithm. For example, the complexity of the RSR algorithm is $O(n)$ using buddy allocation on an n dimensional hypercube.

3.4 Performance Evaluation of the RSR Scheme

3.4.1 Simulation Environment

Two workload models have been used in [29, 69] to characterize the jobs in a parallel computer system, the uncorrelated workload and the correlated workload. Uncorrelated workload model assumes that the work demand of a job is not related to its degree of parallelism. Therefore, jobs executed on more processors are likely to have shorter execution time. This workload model agrees with the assumption used in the Amdahl's law [67] for speedup. Correlated workload model assumes that the work demand of a job depends on the number of processors used for its execution. Workload of a job can be scaled up when more processors are allocated for its execution. This assumption is justified by the Gustafson's law [70] which states that many scientific computation can be scaled up to obtain higher accuracy when more processors are used. RSR allocation reduces the number of processors allocated to a job and increases the execution time of

the job. It is therefore safe for us to adopt the correlated workload in the simulation of RSR scheme because the increase of execution time in correlated workload is more significant than that with uncorrelated workload assumption. The RSR scheme should perform better or equally well when the uncorrelated workload is assumed.

Table 3.1 Default Simulation Parameters.

System Size (Mesh)	32x32
Service Rate	0.2
Number of Completions per Simulation	10,000
Jobs bypassed before data collection	1,000

Table 3.1 lists the default system parameters used in the simulations. This set of parameters is also used in the rest of this dissertation unless otherwise noted. The system simulated is a 32×32 mesh. The service rate for jobs is 0.2 thus the mean service time is equal to 5. Each simulation collects the data for 10,000 job completions with the first 1,000 jobs bypassed to avoid premature data. Table 3.2 shows the parameters assumed for the arrived jobs. Jobs are assumed to arrive in a Poisson process with the service time assumed to be exponentially distributed. To demonstrate the performance of different allocation schemes in a dynamic environment, different arrival rates are simulated. The arrival rate is calculated from the *traffic ratio* which is defined as the ratio of arrival rate to service rate. Size of a job in each dimension follows the same distribution independently. Two different distributions for the submesh sizes are simulated. The uniform distribution assumes the side-lengths of a job range from 1 to 32 with equal probabilities. We also simulate truncated normal distribution for the side-lengths of a job in which the mean is set to 16.5 with a variance of 6.6. The results presented throughout the dissertation are collected over repeated iterations of the simulation to reduce the possible margin of error.

To demonstrate the flexibility of the RSR scheme and to compare with the limit allocation which is proposed for the hypercubes, the RSR scheme is also simulated for

Table 3.2 Job Characteristics in Simulation.

Job Arrival Process	Poisson
Uniform Distribution	$P_i = \frac{1}{32}, i = 1 \ldots 32$
Truncated Normal Distribution	Mean16.5 with Variance6.6

an 8-cube system. Job size in the hypercube simulation is assumed either uniformly or normally distributed between 0 and 7 dimensions. Because an 8-cube job would have to wait for the completion of all other jobs executing in the system for any allocation scheme, it will have the same effect for any allocation scheme. Therefore, we have not include 8-cube requests in our simulations. The probability for any request between a 0 and a 7-cube is equals to 1/8 for the uniform distribution. Jobs with normally distributed dimensions are also simulated for the hypercubes. The normal distribution of the job size is obtained by discretizing the probability of a normal distribution between -2.5σ and $+2.5\sigma$ of its mean. The probability obtained is normalized to one to include the probability outside this region. The resulting probabilities for different job sizes in the 8-cube system are *(p0 = p7 = 0.025, p1 = p6 = 0.076, p2 = p5 = 0.162, p3 = p4 = 0.237).*

The metrics of interest in these simulations are the average turnaround time of jobs to reflect the system performance from the users' perspective. The turnaround time of a job is the time between its submission and completion. Systems with a low average turnaround time is expected to complete a task faster than systems with a high average turnaround time and also have higher throughput in general. Another important indication of the system performance is the traffic ratio at which the system becomes saturated. When a system becomes saturated and cannot handle the load, the average turnaround time of the system increases rapidly. This can be observed from the results of the average turnaround time.

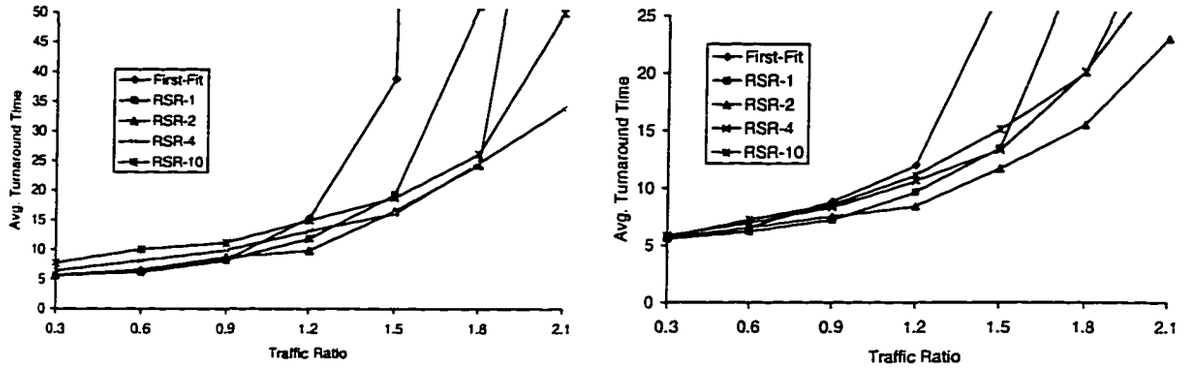
3.4.2 Simulation Results

3.4.2.1 Average Turnaround Time for Jobs with RSR

The average turnaround time of the RSR scheme is compared with the other allocation schemes in Figure 3.3 to 3.5. The RSR scheme has shorter average turnaround time for jobs in either mesh or hypercube-based systems. Using RSR, a job is executed as early as the size reduction restriction allows. Therefore, it is less likely to block other jobs. The improvement is more visible under high system load where the blocking effect is more pronounced, and cause the average turnaround time to increase substantially. With RSR, this effect is reduced and therefore the average turnaround time is also reduced.

Another observation made from these results is the tradeoff between the larger operational range and the lower average turnaround time. When the system load is high, a system needs to accommodate as many jobs as possible in order to avoid saturation. Allowing more size reduction makes this possible. However, as noticed in the RSR scheme, allowing smaller number of size reductions provides shorter turnaround time under low to medium load. This is because the allocations allowing more size reductions tend to reduce the size of a job more often. As fragmentation is not serious under these loads, execution time is the dominant factor in the average turnaround time. Therefore, allowing less number of size reduction avoids the unnecessary job size reduction and provides a better performance under such loads. On the other hand, the system performance improves rapidly with a small number of size reductions. The performance improvement of allowing more size reduction is not significant at low to medium load. Higher number of size reduction, although allows the mesh to be operated with a higher system load, also have very high turnaround time at high traffic ratio because many jobs encounter severe size-reduction. The execution time for a job requiring 1024 nodes may be increased 1024 times if it is folded down to a single node. Moreover, the size-reduction

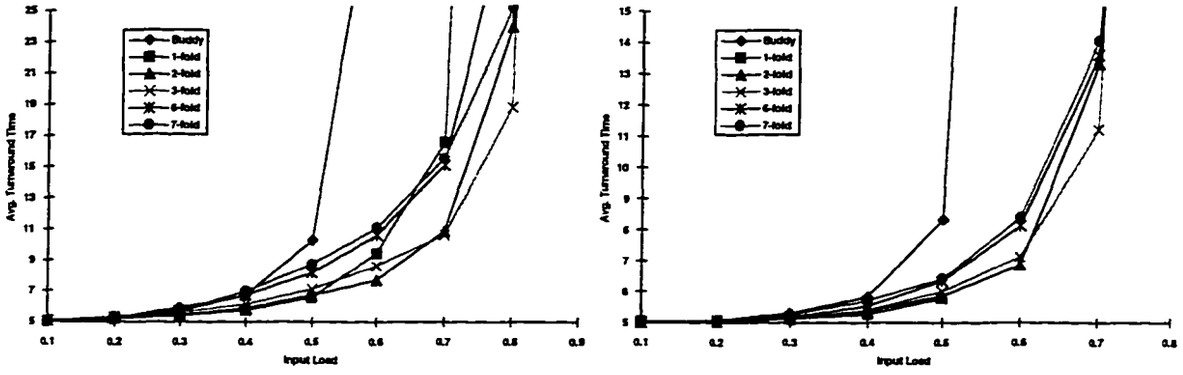
may be restricted by the memory space available at the nodes. In our experiments, we have assumed sufficient memory space at the nodes to demonstrate the effect of various degree of size reductions. It is preferable to allow a small number of size reductions, such as two or four, to improve the system performance while avoiding unnecessary overhead caused by large number of size reductions.



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

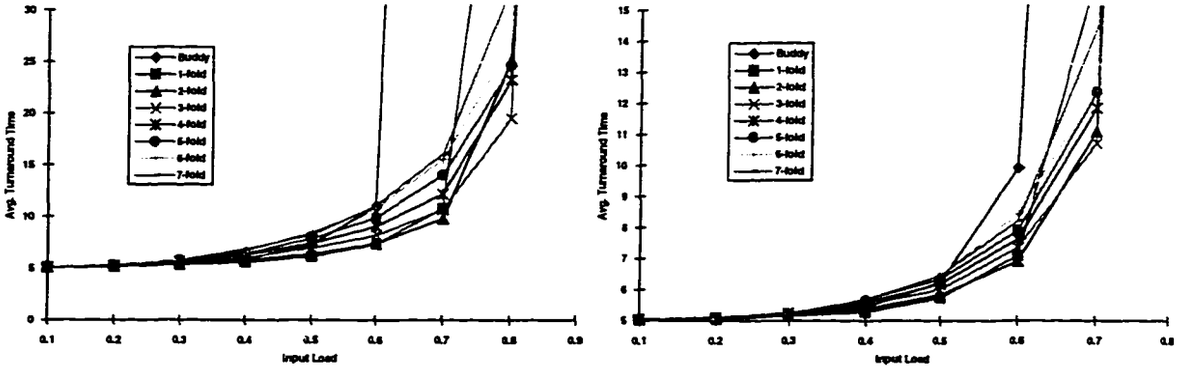
Figure 3.3 Average Turnaround Time for RSR Schemes in a 32 × 32 Mesh.

The RSR scheme is also simulated with the buddy allocation algorithm for the hypercube systems. The results are shown in Figures 3.4. We also include a set of results for the hypercube system in Figure 3.5 where truncated normal execution time is assumed. The simulation for hypercube is conducted with varying system load. System load is roughly equal to the system utilization before system saturation. The arrival rate λ can be calculated for a given system load as $\frac{\text{system size}}{\text{mean job size} \times \text{mean service time}} \times \text{system load}$. Similar observation regarding the operational range of the system and the average turnaround time of jobs can be made. The RSR scheme improves the operational range of the system dramatically. The average turnaround time of jobs is also reduced with the help of the RSR scheme. The turnaround time of the RSR schemes allowing less numbers of size reductions have shorter turnaround time than those allowing larger numbers of size reductions under low to medium load. Again this is caused by the penalty of applying size-reduction to jobs.



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

Figure 3.4 Average Turnaround Time for Different Allocation Algorithms in an 8-cube Using the Buddy Allocation with Exponentially Distributed Service Time.



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

Figure 3.5 Average Turnaround Time for Different Allocation Algorithms in an 8-cube Using the Buddy Allocation with Truncated Normal Service Time Distribution.

The RSR scheme improves the performance of both the mesh and hypercube systems at especially high traffic load. This is because of the blocking effect associated with the FCFS discipline. Under the high system load, a job that cannot be allocated immediately is more likely to block other jobs. Because of the FCFS discipline, all the succeeding jobs will have to wait for this job before they can get into the system for execution. Using the RSR, a job is executed as early as the size reduction restriction allows. Therefore, it is less likely to block other jobs. The fragmented processors are also utilized to execute the folded jobs. The turnaround time of the RSR schemes are hence shorter.

Another observation made from these results is the tradeoff between the larger operational range and the lower average turnaround time. When the system load is high, a system needs to accommodate as many jobs as possible in order to avoid saturation. Allowing more size reduction makes this possible. However, as noticed in all the algorithms simulated, the allocations allowing smaller number of size reductions provides shorter turnaround time under low to medium load. This is because the allocations allowing more size reductions tend to reduce the size of a job more often. As the blocking effect is not serious under these loads, the execution time is the dominant factor of the average turnaround time. Therefore, allowing less number of size reduction avoids the unnecessary job size reduction and provide a better performance under such loads. On the other hand, the system performance is improved rapidly with a small number of size reductions. The performance improvement of allowing more size reduction is not significant at low to medium load. Higher number of size reduction, although allows the system to be operated under a higher system load, also have higher turnaround time when the system is heavily loaded. This is because many jobs encounter sever size-reduction under this kind of system load. For example, the execution time for a job requiring 1024 nodes may become 1024 times of its original execution time if it is folded down to a single node. Moreover, the size-reduction may be restricted by the memory space available at the nodes. In our experiments, we have assumed sufficient memory

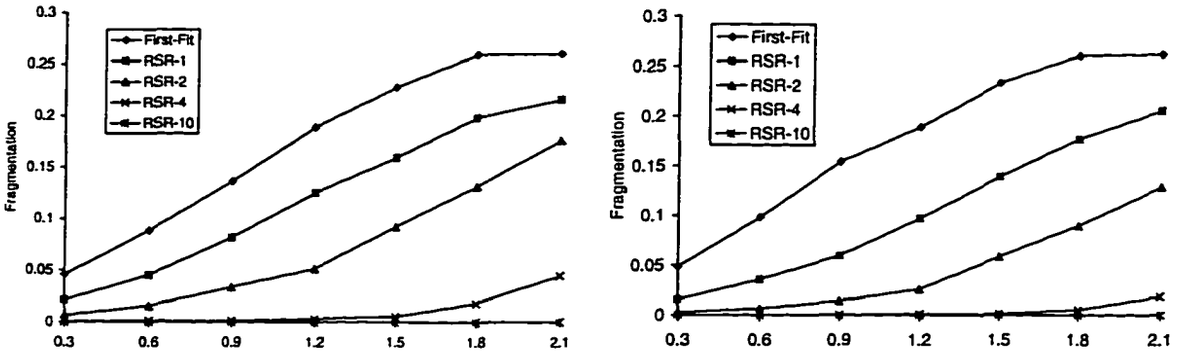
space at the nodes to demonstrate the effect of various degrees of size reductions. It is preferable to allow a small number of size reductions, such as two or four, to improve the system performance while avoiding unnecessary overhead caused by large number of size reductions.

3.4.2.2 Fragmentation of RSR Schemes

As fragmentation is the major cause of system underutilization, we compare the fragmentation of several RSR schemes in the mesh. To capture the effect of processor allocation schemes on the system fragmentation, we define a quantitative measure of fragmentation. To include all kinds of fragmentation, it is defined as the summation of ratio of available processors to the total number of processors at each allocation failure divided by the total number of allocation attempt. Therefore, fragmentation is a function of the system load and efficiency of the allocation scheme. At the same traffic load, a better allocation scheme should produce less fragmentation. Figure 3.6 illustrates the fragmentation of RSR schemes with different restrictions on size-reductions. Results are shown for uniform and normal job size distributions. Both the distributions show similar trend in the results. As the maximum size-reduction, t increases, the fragmentation is reduced. Fragmentation is reduced in two ways. First, the physical fragmentation is reduced because scattered nodes can be utilized to execute size-reduced jobs. Second, the insufficient resource fragmentation is also alleviated because jobs are able to run with smaller number of processors. In the extreme case of *RSR-10*, any job can be executed on a single node. There is no fragmentation of any kind in this case.

3.4.3 Comparing with the Limit Allocation

The limit allocation is claimed to be the most efficient processor management strategy for the hypercubes and has many conceptual similarities with the RSR schemes. Therefore, we compare the RSR scheme with the limit allocation for the hypercube sys-



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

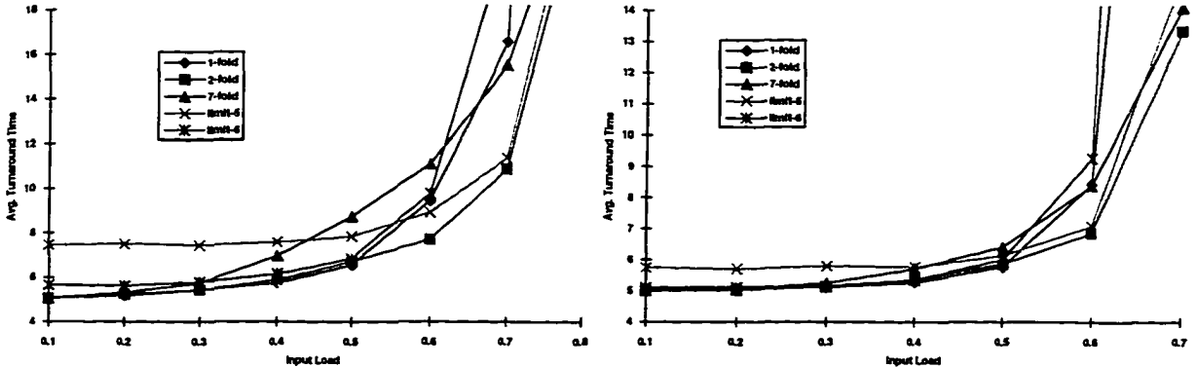
Figure 3.6 Fragmentation vs. Traffic Ratio for RSR Schemes in a 32 × 32 Mesh.

tem. Both schemes use the buddy allocation as the basic allocation algorithm. Figure 3.7 shows the mean response time for the limit allocations and the RSR schemes.

It is important to point out the similarity between the limit allocation and the RSR allocation scheme before making comparisons. Both approaches allow the reduction of job sizes. The *limit-k* allocation reduces all jobs larger than a k-cube to k-cube. The *RSR-t* allocation allows the size of a job to be reduced at most t times. Therefore, it is fair to compare the allocation from different family with the same maximum number of folding. In the 8-cube system we simulated, limit-0 and RSR-7, limit-5 and RSR-2, limit-6 and RSR-1 are comparable schemes. Notice that as discussed in Section 3.3, the RSR-7 allocation is equivalent to the *greedy limit* allocation. Since it is pointed out in [42] that the greedy allocation always outperforms or equally well than the *average limit* allocations, we did not include the more complex average limit allocation in this comparison.

Figure 3.7 clearly shows that between comparing schemes from the two families, the RSR methods perform better than the corresponding limit schemes. The limit-0 allocation is not even shown in both figures because it has a high turnaround time at

around 160 for the uniformly distributed job and at around 60 for the normal-distributed job. All the limit-k allocations have relatively poor performance at low to medium load. This is because of the underutilization problem discussed in Section 3.4.2.1. Because larger jobs are forced to run in a smaller cube in limit allocation regardless of the system load, the larger jobs always have longer execution time. In the extreme case such as limit-0, a job initially requesting for a 7-cube would spend 128 times of the execution time when it is granted a 7-cube. Hence, the mean response time is increased. For the RSR allocations, a job is folded only when necessary. Under a low input load, a job almost never gets its size reduced and is executed at the full speed. When the load increased, more jobs get folded. But since we restrict the number of times size reduction can be applied to a job, increase on the execution time is limited.



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

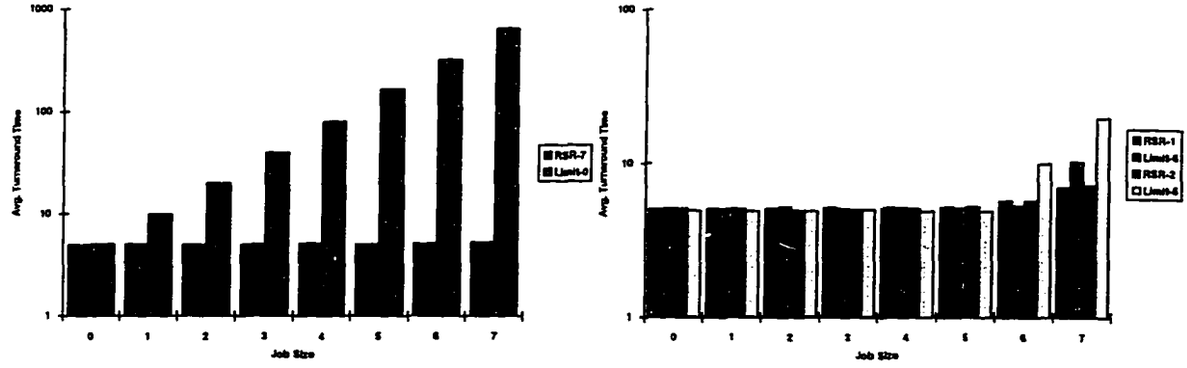
Figure 3.7 RSR vs. Limit in an 8-cube. (Exponential Service Time Distribution)

Along with the better performance, the RSR scheme also provides better fairness toward job of different size than the limit allocation. Figures 3.8 and 3.9 show the average turnaround time for jobs of different sizes at medium (0.3) and high (0.7) system loads. The extreme comparisons between the RSR-7 and the limit-0 allocation are illustrated in part (a) of both figures. The RSR-1 and RSR-2 are compared with the limit-6 and limit-5, correspondingly. The turnaround time is plotted in logarithmic scale because of

the big difference on the response time for the limit-k allocations. Limit-k family forces all larger jobs to be executed on smaller cubes, therefore the execution time increases linearly for jobs requiring cubes larger than the limit. This is directly reflected in the response time for system under low to medium load in which the execution time is the dominant factor. The RSR methods, on the other hand, only fold the jobs when they have to. Under the low to medium input load, jobs are seldom folded, hence the response time remains almost constant for different job sizes. Figure 3.8 depicts this observation.

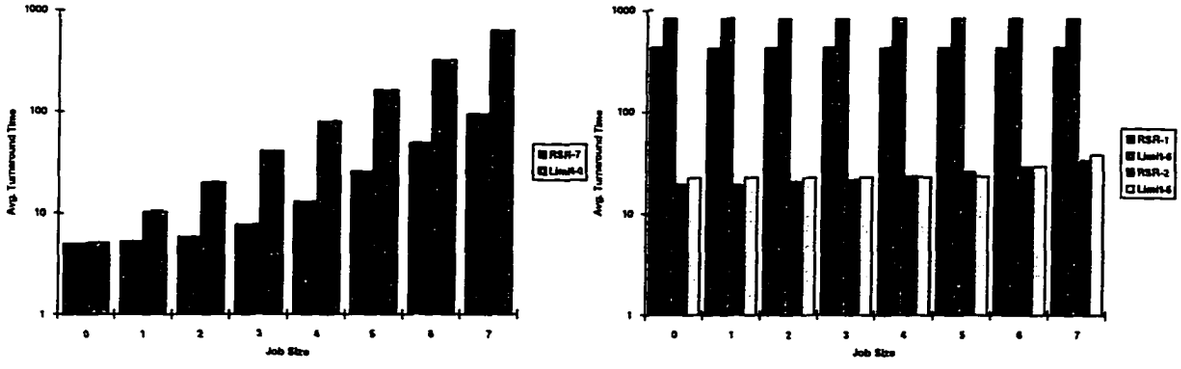
For system under high load, execution time is no longer the dominant factor. This is demonstrated in Figure 3.9. Limit-6 and RSR-1 have close response time for all job sizes. This is because the longer queuing delay jobs experienced in such load outweighs the execution time. RSR-2 and RSR-7 started to treat jobs unfairly at such a load. Jobs requiring larger subcubes have higher turnaround time than smaller jobs. Size reduction happens more often under heavy load. Since larger jobs are more likely to be affected by the fragmentation, their sizes are more likely to be reduced. With a higher size reduction restriction, larger jobs could be folded more than once. The reduction of the job sizes increases the execution time of the larger jobs. The unfair treatment of limit-0 and limit-5 is still visible under this load because the queuing delay still does not outweigh the turnaround time for these two allocations yet.

As indicated in [42], the *average limit* improves the performance of the multicomputers less efficiently than the *greedy limit*. The authors claims the *greedy limit* to be the most efficient processor management strategy for the hypercube systems. The greedy limit is a variant of our proposed RSR allocation scheme when the restriction of size reduction allows all jobs to be executed on a single node when necessary. However, our results in Section 3.4.2.1 points out that allowing too much size reduction results in unnecessary folding of the jobs under low to medium load. The performance of the system under such load is sacrificed with the excessive size reduction. Therefore, it is not desirable to use the greedy limit allocation. In the normal operations of low to medium



(a) (b)

Figure 3.8 Fairness Comparison between the Limit and the RSR Scheme at a Medium System Load (0.3) for an 8-cube System.



(a) (b)

Figure 3.9 Fairness Comparison between the Limit and the RSR Scheme at a High System Load (0.7) for an 8-cube system.

load, the RSR schemes allowing the same number of size reduction outperforms the corresponding limit-k allocations. The RSR schemes also provide much fairer treatment to jobs of different sizes. Therefore, we conclude that the RSR scheme is better than the limit allocations from the comparisons.

3.5 Discussion

The RSR scheme reduces the size of a job which may not be possible for some applications. Memory threshold is another problem associated with RSR scheme. When

a job i allocated to a smaller submesh, the storage requirement on every nodes assigned is increased. It is possible to exhaust the available memory by size-reduction and hence results in heavy swapping of memory.

Judicious selection of maximum size reduction for RSR is important. Our results suggest to use small values for this because the performance gains may be offset by the tradeoffs with large number of size-reduction. In RSR, the tradeoff is the increased execution time. By using small values for these parameters, the above problems of memory space and communication overheads can also be avoided. RSR can be implemented to allow individual jobs flexibility in specifying number of size reductions. With this flexibility, a job requiring a large memory space can request the allocator to not fold it.

4 MEASURING THE EFFECT OF PROCESSOR ALLOCATION ON COMMUNICATION LATENCY

4.1 Introduction

One possible processor management approach to improve the performance of multicomputer is by allocating processors non-contiguously. The goal of this approach is to eliminate the fragmentation problem. Non-contiguous allocation algorithms can be classified as totally non-contiguous or partially non-contiguous. In a totally non-contiguous allocation schemes, allocation is purely based on the availability of nodes while in partial non-contiguous allocation, certain degree of contiguity is maintained among the allocated nodes. Through simulation studies, the non-contiguous allocation schemes are shown to reduce fragmentation effectively. Queuing delay in a system using non-contiguous allocation is expected to be less because of the absence of fragmentation. However, the communication latency of a job allocated non-contiguously is increased because of the increased distance of the communication path and the contention of messages in the interconnection networks. There is also a possibility that the interprocess interference may saturate the interconnection network and cause all jobs to experience very high message transfer latency.

A message goes through three stages during its lifetime, the preparation stage, network stage, and the consumption stage. Preparation and consumption stages are the phases in which a processor processes the message to be sent and receive the message, respectively. The network stage involves the actual time spent while a message travels

through the interconnection network. A message experiences various delays in these three stages. In the preparation stage, message may encounter queuing latency if the outgoing links are occupied by previous messages or by messages that travels through the links. A message may also have to wait in the receiving node's buffer before it can be consumed if there are other messages destined to the same nodes that have not been consumed yet. The network stage contributes to the communication latency in several ways depending on the switching techniques implemented in the interconnection network. For instance, if the interconnection network uses circuit switching, then the network stage includes setting up and terminating the connection plus the real transmission along the established path. If store-and-forward switching is used, this stage include the entire duration of a packet being stored and forwarded along the path. When wormhole routing is used, then this stage includes the time for the worm to go through from source to destination and all consequent blocking encountered by the worm.

Non-contiguous allocation can have a dramatic effect on the message latency. First, non-contiguous allocation means longer communication paths between allocated nodes. Liu et al. [25] argue that the longer communication paths play an insignificant role in the communication latency if wormhole routing technique is implemented. However, many systems are not built with the wormhole routed interconnection networks. Earlier system such as nCUBE 1 and iPSC-1 use the store-and-forward switching mechanism. Even with wormhole switching, a longer path means a higher possibility of blocking the worms and the blocking may cause a higher communication latency. The most important issues about non-contiguous allocation is the contention on the communication links among messages. This contention can cause the message to be blocked in the interconnection network or cause it to be buffered at the sending node. These extra delay may cause the interconnection network to be saturated and therefore causes the application waiting indefinitely.

To validate the feasibility of non-contiguous allocation schemes, one has to show that

the increasing communication latency is insignificant compared to the decrease in the queuing latency. The communication latency in a wormhole-switched network has been studied in [56]–[64]. Communication latency is studied specifically with job allocations in [25, 58, 59]. In [58] and [59], communication latency of allocated jobs in a multicomputer are studied based on simulations. Both studies indicate the increase of message latency is not significant when scattered allocation is applied. In addition, [58] also considers the effect of synchronization which can be the major problem in a system with high variance on communication latency. Liu et al. [25] conducted both experimental and simulation study on the message passing latency in support of their non-contiguous allocation algorithms. Their simulation study agrees with the above observations. However, in their experiment study conducted on Intel Paragon, an interesting result is shown when the a more efficient OS (SUNMOS) is used. In that experiment, the communication latency increase almost linearly with the number of contending messages. This is in contrast to the simulation studies.

Lack of information about the communication latency in a non-contiguously allocated environment limits the applicability of the bounding estimation. Therefore, we design a set of experiment on a nCube 2 multicomputer and measure the actual communication latency experienced by jobs running on such a system. The significance of this research is its realistic nature. Most of previous studies on communication latency regarding processor allocation compromise the reality by making assumptions to simplify the analytical modeling or are done through simulations. To our knowledge, no one has attempted to perform a realistic measurement on a running system as this work presents. Our results provide important information for designers of multicomputer systems for choosing a better processor management strategy.

4.2 Experiment Setup

An experiment on an nCUBE-2 multicomputer is conducted to evaluate the interprocessor communication latency and study the effect of various processor allocation schemes on the communication latency. Three communication models based on commonly used applications are formulated to run with various allocations. We start the discussion on our experiment design with these communication models, followed by the experimental environment and our approach of measurement.

4.2.1 Communication Models

A variety of jobs with different communication patterns are executed on multicomputers in scientific computation environments. Based on the communication pattern commonly seen in parallel applications, we selected a few communication models for this study. Study on the other patterns are in progress. The communication patterns studied here are *nearest neighbor, polling,* and *random.* The details of these communication patterns are described next.

- *Nearest Neighbor:* Every node communicates with its nearest neighbor for information exchange. The communication path, in this case, is usually short and optimized. This is a typical communication pattern used in matrix manipulations.

- *Polling:* A central node sends a message to all the other nodes running the same process. After receiving the message, the receiving node sends back a message to the polling node. The central node can keep on sending without waiting for the reply message until it finishes a round. The central node starts the next round of polling after receiving the reply from all the nodes. This pattern is commonly seen in programs where jobs are distributed to the processors and one processor remains in charge of distributing operands and collecting results. This pattern creates a

hot spot in the central node with which every node attempt to communicate. An example of such applications is the exhaustive search of complex data structure when load has to be dynamically balanced among the nodes. Polling is also used for barrier synchronization and cache invalidations.

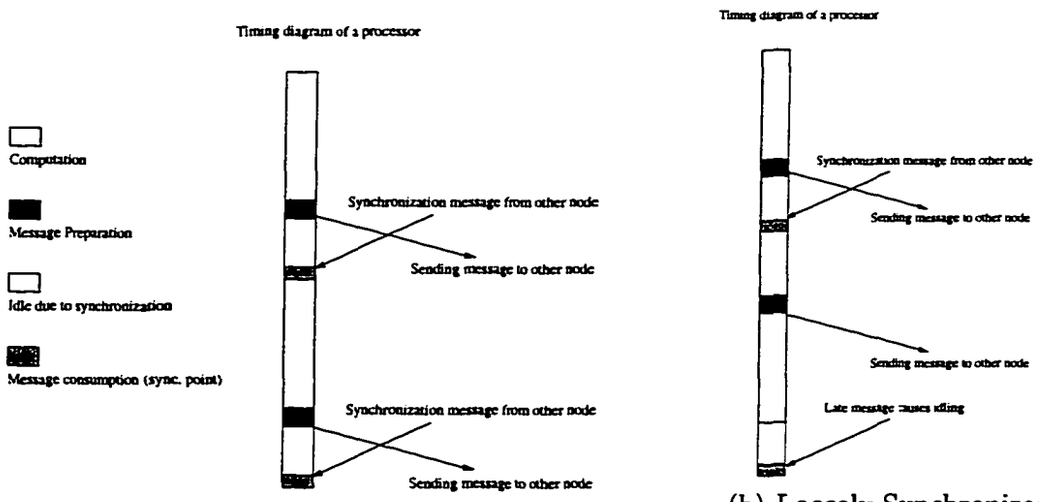
- *Random:* Every node picks its destination independently and sends out a message. The receiving node sends back an acknowledgment and the sending nodes have to wait for the reply before continuing its operation. Updating and managing a large database system is an application that normally uses such a communication pattern.

In real applications, a mixture of several communication patterns may be observed. Although our experiment evaluates the effect of processor allocation on communication latency for these communication patterns separately, it should provide a reasonable understanding of the latency incurred due to various processor allocation schemes.

The traffic patterns have to be mapped to the topology of the system to optimize the communication path. Polling and random patterns do not have an obvious optimized mapping because of the uniformness of distribution of the destination nodes. For the nearest neighbor pattern, we use a gray code mapping so that only the nodes which have a Hamming distance of 1 intercommunicate. Hence the communication distance is reduced to 1 hop for every message. In addition, to simplify the implementation of our experiment, we implement the nearest neighbor pattern as a ring in which every processor sends its message to only one of its neighbors and thus the allocated processors form a logical ring.

Communication latency plays an important role in the average execution time of jobs, especially when synchronization among nodes are considered. Two possible synchronization scenarios occur for different types of messages in parallel applications. In a *tightly synchronized* scenario, a synchronizing message is expected when a message is

sent to another node. A synchronization message may be an acknowledgment from the receiving node or some information sent by another node. In the tightly synchronized communication, the synchronizing message contains critical information for the destined node to continue its operation. The execution of the program does not advance until it receives its expected synchronizing message. The tightly synchronized communication is commonly seen in parallel applications such as matrix manipulations in which matrices are divided into submatrices. To complete the manipulations, each node carries out operations over the local data and the data need to be exchanged. The second type of communication model is loosely synchronized. In this case, a synchronizing message is expected but the processor waiting for this message is allowed to continue its operation for a certain period of time before it gets stalled. The polling pattern in our communication model is an example of loosely synchronized communication. The central node which polls others nodes can continue its polling without waiting for an immediate reply from the polled nodes. The central node stops only between rounds of polling to collect the response from all the polled nodes. Both types of synchronization models are illustrated in Figure 4.1.



(a) Tightly Synchronized Communication

(b) Loosely Synchronized Communication

Figure 4.1 Synchronization Models.

In Figure 4.1(a), messages are exchanged between the specific processor and other processors in a tightly synchronized manner. After sending out its message, the processor has to wait for the corresponding synchronization message to arrive before it can continue its computation. Before the arrival of the synchronization message, the processor idles. Figure 4.1(b) shows a loosely synchronized scenario. After sending out a message, the processor continues on with its computation. If the synchronization message arrives in time before the next round of synchronization, the processor does not have to idle. However, if the second synchronization message misses its synchronization point, it causes the processor to idle before its arrival.

4.2.2 System Environment

We used an nCUBE-2 system as our test-bed, which has 128 processors in a seven dimensional hypercube interconnect topology. Each node is proprietary designed running at 20 MHz. A proprietary operating system called Vertex is used.

Routing in nCUBE-2 is based on a deterministic model and is implemented in hardware. The maximum distance between any two nodes equals to the dimension of the subcube that encloses the two nodes. Wormhole switching technique is used to propagate messages between the nodes. Wormhole switching shortens the communication latency and reduces the amount of buffer required in the switching hardware compared to packet switching and virtual cut-through. Promoters of non-contiguous allocation schemes suggest that the use of wormhole switching makes the communication latency insensitive to the distance. The adoption of non-contiguous allocation may otherwise not be feasible if a significant penalty on communication latency is incurred.

Our experiment is designed to study the effect of processor allocation and synchronization on the communication latency. Two things are particularly of interest, the effect of contention caused by messages generated by different processes and the contention caused by messages generated by the same process. Instead of a simulation-based study

as done by previous researchers, we have targeted on measuring real communication latency for jobs running on an nCUBE-2 system. There were several problems that needed to be overcome for this experimentation. First, it is necessary to devise a mechanism such that the measurement of the latency and the collection of other information does not affect the execution of the process. Measuring the communication latency on a node executing a program may introduce extra workload on the processor. This overhead should be kept at a minimum. Second, the current version of the nCUBE-2 OS, Vertex, does not support the feature of non-contiguous allocation. One of the main purpose of our study was to evaluate the effect of non-contiguous allocation. As this option was not supported by the default operating system, we had to design a mechanism to handle the scenario. Third, processors are only allowed to communicate with other processors within an allocated subcube. Messages are not allowed to travel out of the boundary of an allocated subcube. It is therefore very difficult to measure the effect of contention on the communication links due to messages generated by different processes. One of our intentions was to study effect of contention between different processes, so we have to somehow enforce such an interference to examine its effect.

To handle the problems mentioned above, the communication models are implemented on an emulator program. To overcome the lack of support for non-contiguous allocation in the nCUBE-2, these communication models are programmed as pseudo jobs contained in a large program. This large program runs as a *hypercube emulator*. A large subcube is acquired for running the hypercube emulator and pseudo jobs were allocated to any set of processors within the allocated large subcube. Upon loading, each processor is given its own set of parameters which indicates its communication partners and the characteristics of the pseudo job it executes. It then participates in the execution of the pseudo job according to the given parameters. This approach allows flexibility in various allocation options and job characteristics such as the non-contiguous allocation, communication frequencies, and communication patterns.

Running a profiling tool on top of a program to monitor its execution and communication introduces overhead to the processor. If the overhead is significant, the measured data from the monitored process differs from their actual values. Since the communication latency is the main interest of this experiment, our implementation of pseudo jobs hides the data collection and calculation in the computation phase of the pseudo jobs. Figure 4.2 shows an example of the flowchart of the execution of a pseudo job. The flow chart shown is for the jobs that are tightly synchronized. A similar chart is used for loosely synchronized tasks. This implementation also provides the flexibility of varying the communication frequency or the computation to communication ratio.

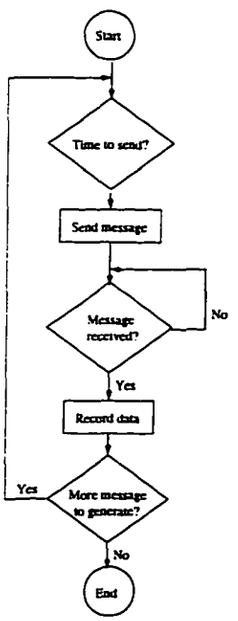


Figure 4.2 Flowchart of a Job Execution Cycle under Tight Synchronization.

4.3 Results

The emulator program is executed to evaluate the message latency with respect to several variables. In case of polling, the central node polls each of the other processors for 1000 times before it is assumed to be completed. The nearest neighbor and the random

jobs are assumed completed when every node has sent 1000 messages to its destination. Between generation of messages, a delay interval is inserted to model the behavior of a parallel application. In a real application, communication is done between phases of computations. The amount of computation affects the frequency of communication. The delay interval between message generations in our emulator models the computation phase of a processor. A shorter delay interval represents a job with higher communication demands and thus requires more frequent communications. A longer delay interval represents a computation-intensive job. Two message sizes, 64 and 8000 bytes, are used in our experiment. Additional information such as the destination address are included in the packets for transmission making the actual packet size a little longer than 64 and 8000 bytes. The results shown in this section is the average taken over several repetitions of the experiment. The latency or delay units shown in all the graphs are in microseconds.

In the following subsections, we have reported results obtained through four different types of studies. First, we evaluate the latency variation with respect to the delay interval. We also study the latency variation for different sizes of jobs thereby investigating the effect of the communication path length on the latency. Second, we examine the effect of the geometry (relative locations) of the allocated processors on the communication latency. Third, the effect of interprocessor interference is studied when a message belonging to one task may encounter messages belonging to different tasks along its path. Fourth, we analyze the effect of different types of processor allocation on the execution time of a task.

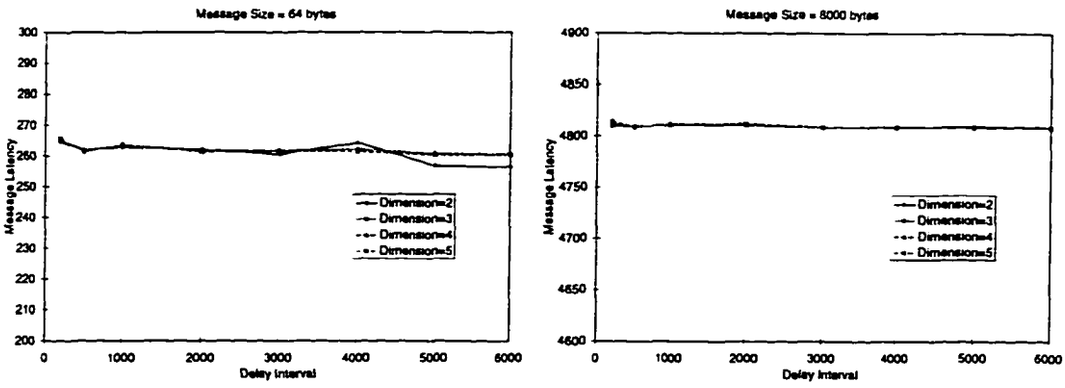
4.3.1 Effect of Communication Frequency and Path Length

Figures 4.3, 4.4, and 4.5 illustrate the relationship between job size and the message latency with respect to the delay interval. The delay interval is the time between which a node receives its synchronization message and generates its next message. It is worth

noticing that the message latency is almost constant throughout the delay intervals. Most researchers study the relationship between MTBT (Mean Time Between Transmission) and the message latency in wormhole routed networks and show a different picture than that depicted in our results. The curves shown in such studies usually rises up very fast after a certain traffic ratio. MTBT and delay interval both represent the communication demand of a job with one major difference. When MTBT is used instead of delay interval, the message latency is relatively constant except when the MTBT is less than some certain value. However, our result of communication latency remains almost constant for all delay intervals. This is because of the synchronization constraints. As no new messages are generated during the delay interval because of the computation and synchronization, messages seldom get queued and the communication latency is incurred because of the transmission delay of the packets. Studies using MTBT usually assume a random variable for the interarrival time of messages. A node can send out many messages before its earlier messages are consumed by the receiving nodes. Because of this unrealistic assumption that requires no synchronization, messages may be queued indefinitely at the receiving nodes. The queuing delay makes the communication latency intolerable for systems with low MTBT. However, in a real application, it is unlikely for a node to send out messages in such an unrestricted fashion.

In Figure 4.3, no significant difference is observed between jobs of different dimensions. This is because the nearest neighbor traffic pattern has the shortest communication path (1 hop each) and does not have any interference between messages. We have assumed tight synchronization for the nearest neighbor communication. So no messages are generated during the delay interval. The average latency remains almost constant as the message are never queued at the destination nodes.

The result for the polling traffic shown in Figure 4.4 depicts an increase in message latency when the dimension of a job is increased. This is believed to be caused by the contention for the communication paths among the polling messages. In a larger cube,

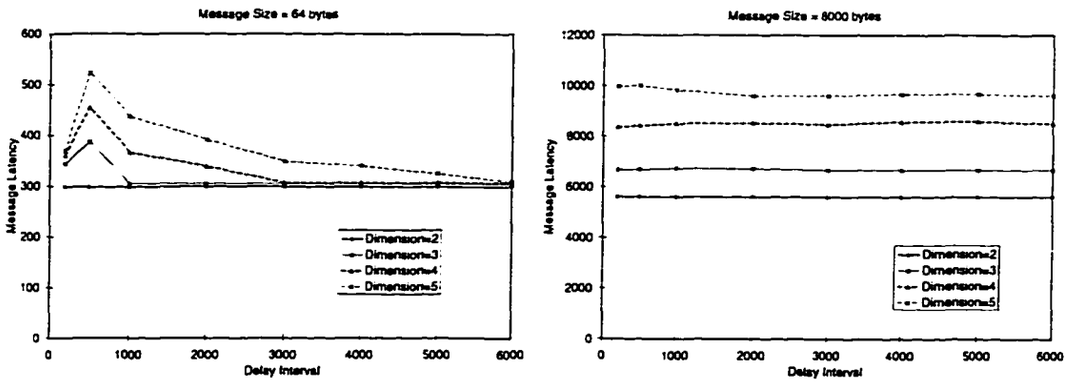


(a) (b)

Figure 4.3 Message Latency vs. Job Size for the Nearest Neighbor Pattern.

not only the communication distance is increased, but also the number of processors that have to be polled is increased. The increased number of messages increases the probability of blocking. When the path of a message is held by another message, it has to be queued and therefore incurs delay before its arrival to the destination. This observation contradicts some of the previously reported studies that claim that wormhole routing is completely insensitive to distance. The insensitivity is true only for a contention-free network. The increase in message latency is prominent in Figure 4.4(b) where the message length is high. For short messages (Figure 4.4(a)), the increase is only seen with shorter delay intervals. Because short messages occupy the communication path for a short period of time, it is less likely to cause other messages to be queued. With shorter delay intervals, messages are generated more frequently, thus making it possible for short messages to contend for the routes. With long delay intervals, the communication paths are more likely to be cleared of previous messages and hence the queuing effect is reduced. However, as in the case of nearest-neighbor communication, here also the latency is almost constant for varying delay interval.

Latency curves for random traffic pattern is shown in Figure 4.5. In case of random traffic pattern, higher dimensional jobs have higher message latency. Again, contention

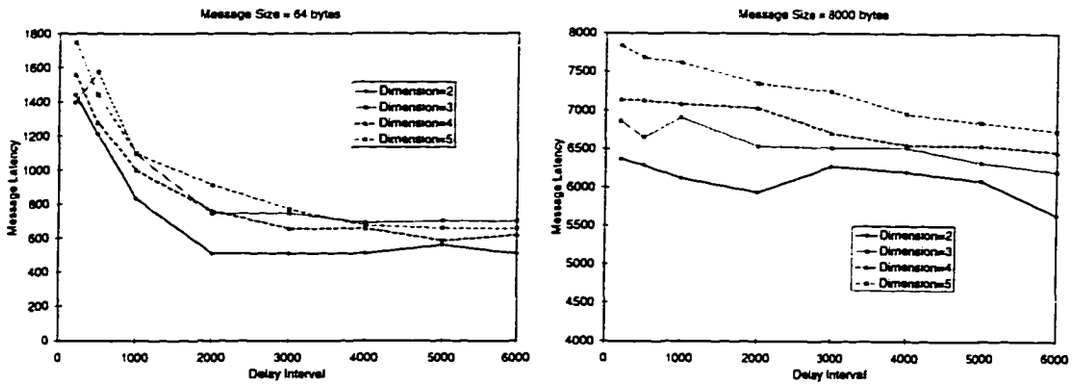


(a) (b)

Figure 4.4 Message Latency vs. Job Size for the Polling Pattern.

on the communication links attributes to this increase. Because the number of nodes in a higher dimensional cube is larger, the number of messages competing for the same route is also higher. Additional contentions at the receiving nodes are also encountered. Because each node picks its destination randomly, it is possible for a processor to receive multiple messages at any instant. Messages that cannot be consumed by the receiving node immediately are queued. The contention on the receiving nodes also attributes to the higher message latency with shorter delay intervals. With a shorter delay interval, the number of messages that can be queued at a node is increased and hence results in an increase in message latency.

It is interesting to compare the results of the polling and the random patterns. When short messages are considered, polling pattern always has shorter message latency than the random pattern. This is because the polling pattern does not have contention on the receiving nodes as the random pattern does. With shorter delay intervals, this contention is more significant and causes a big increase in the message latency. Polling pattern does have a longer message latency for high dimensional jobs because all the route contention involves the central node as the origin of the path. The route contention in the random pattern is distributed among all possible node pairs and therefore is smaller.



(a) (b)

Figure 4.5 Message Latency vs. Job Size for the Random Pattern.

4.3.2 Effect of the Geometry of Allocated Processors

A parallel application is written to utilize the interconnection topology so that the communication pattern and the distance of communication paths can be optimized. There is a possibility for a higher communication latency in a job if the geometry (relative locations) of the nodes allocated to the job is altered. Previously proposed non-contiguous allocation schemes [25] have suggested to preserve partial contiguity. Figure 4.6 shows a simple example of how the contention and the distance is affected by altering the geometry of allocated nodes in a ring architecture. The communication pattern in Figure 4.6(a) is optimized to use the ring topology. Each message takes only one hop to reach its destination. No contention occurs between any two messages. If the allocation of node 1 and node 5 is switched, as shown in Figure 4.6(b), almost all paths become longer. Messages from node 0 to node 1, and messages from node 5 to 0 now have to take five hops instead of one. Contention on all physical links that is shared by more than one communication paths becomes highly likely.

We measure the effect of the geometry of allocated processors on the message latency for the nearest neighbor traffic pattern. Polling and random patterns are not considered here because of their uniformity in communication paths. For the nearest neighbor

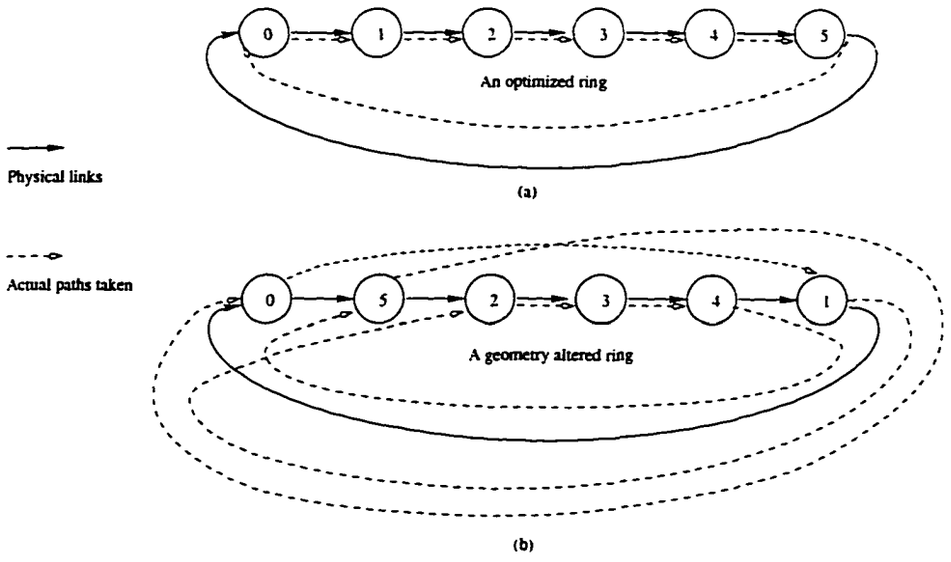


Figure 4.6 Effect of Altering Geometry.

pattern, we alter the geometry of the allocated processors so that the two neighboring nodes always have the highest possible Hamming distance. Figure 4.7 depicts the layout of processors in the two and three dimensional cases with their bit addresses. By arranging the nodes in such a ring, the distance between any two communicating nodes is maximized and hence increases the possibility of contention between messages. Similar rings with maximal Hamming distance between neighboring nodes can be obtained for an n dimensional ring by assigning $n-1$ digits Hamming code to every other node in the ring. The addresses for the rest of nodes can be calculated by inverting the bit address of the preceding nodes.

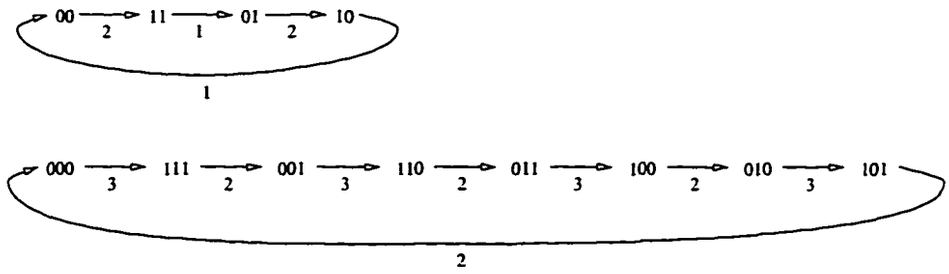
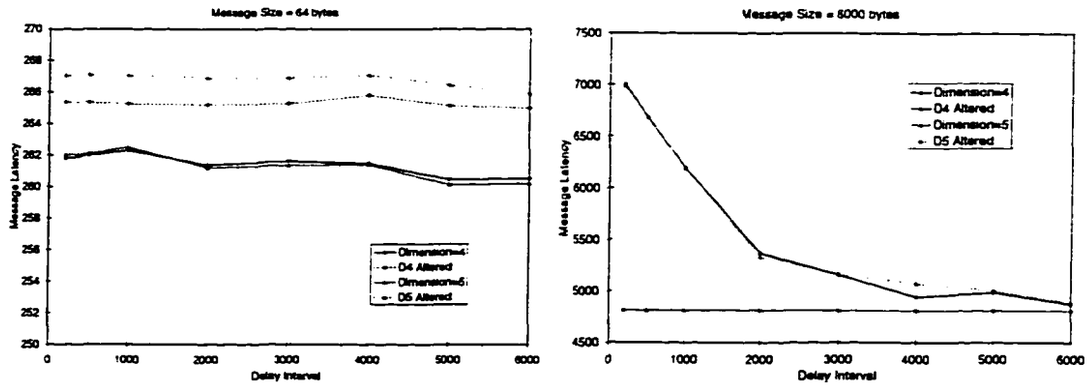


Figure 4.7 Layout of a Geometry-Altered Ring with Addresses.

Figure 4.8 illustrates the results of the experiment on the effect of the geometry of allocated nodes. The message latency in a geometry-altered system is compared to that of a cube with optimized communication path. In the geometry-optimized cube, processors are arranged in a gray-code ring for the nearest neighbor pattern. Message latency remains constant for all delay intervals and for jobs of different dimensions. The geometry-altered allocation produces significantly higher message latency in all the cases studied in our experiments. Higher the dimension of the system, the larger is the penalty of the communication latency due to altered geometry. When the dimension of a job is small, the number of hops that a message has to travel through is limited. For example, the longest route that a message can take in a 2-dimensional cube is only two hops. Chances for messages to interfere with one another is limited and thus results in less penalty on the message latency. With a higher dimensional subcube, the contention on the communication links is more serious and therefore more queuing is observed in the message latency. This queuing delay is less significant when short messages are used and the delay interval is long. The communication link is only occupied by a message for a small amount of time and therefore is less likely to cause messages to be queued in such cases. If long messages and short delay intervals are considered, a significant difference can be observed for jobs of different sizes, as shown in Figure 4.8(b). Therefore, the geometry of allocated processors should be retained as much as possible to reduce the message latency.

4.3.3 Effect of Interference

To measure the effect of interprocess interference, multiple jobs are allocated on the cube that execute simultaneously. Jobs are allocated on the hypercube so that the messages generated by one process have to travel through intermediate nodes that belong to other jobs. There are many possible combinations of jobs and allocations. To simplify the experiment, we allocate two different jobs, each of dimension 4, together in

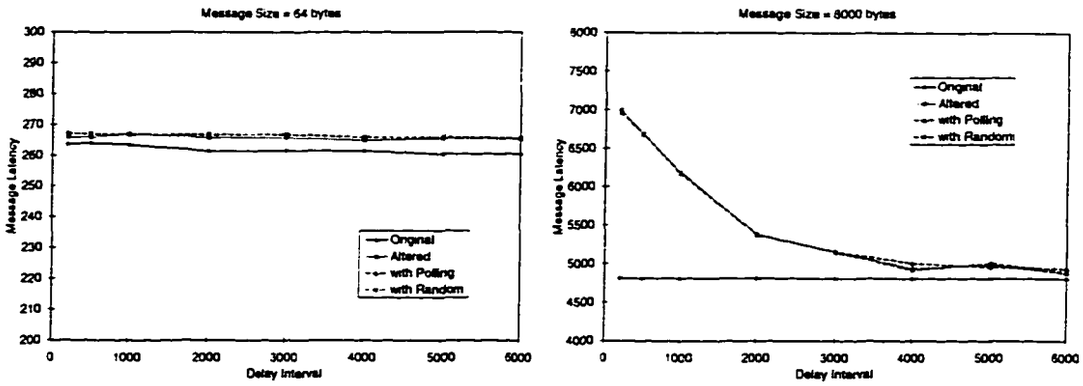


(a) (b)

Figure 4.8 Message Latency in a Geometry-Altered System.

a five-dimensional cube. Each job is allocated on two non-contiguous three-dimensional subcubes to measure the effect of interprocess interference in non-contiguous allocations. Effect of interprocess interference is shown in Figures 4.9, 4.10 and 4.11.

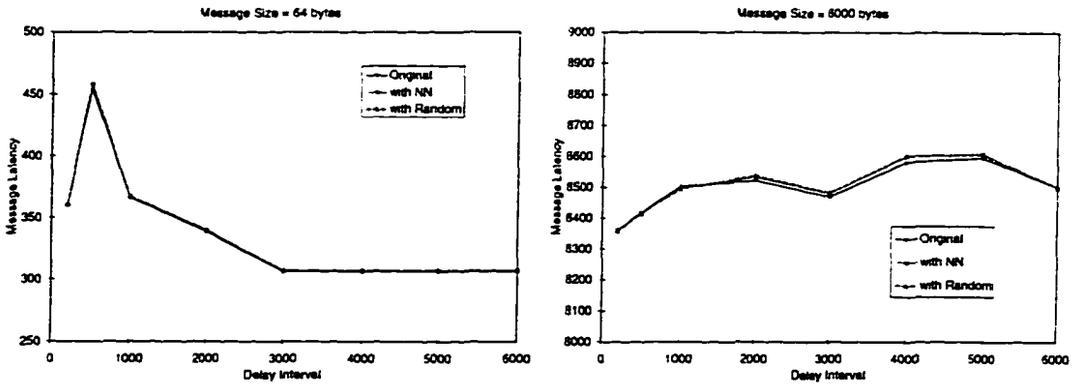
Figure 4.9 shows the message latency for the nearest neighbor pattern job when it is allocated together with the other two traffic patterns. The curve labeled original represents the allocation of an optimized ring using the gray code sequence. The curve labeled altered is the results obtained in the previous subsection when the geometry is altered. The other two curves show the message latency when interprocess interference is introduced. The original allocation provides an optimized communication path for every message causing no contention on the communication path and has the lowest message latency. For both the short and long messages, an increase in the message latency is observed when the geometry of the ring is altered. Interprocess interference caused by allocating another job together does not show significant increase on the message latency. The increase in the message latency for the short message is around 3% for all delay intervals. The increase is more significant for long message with short delay interval. For example, at a delay interval of 200 microseconds, this increase is around 45% compared to that of the optimized allocation.



(a) (b)

Figure 4.9 Message Latency in the Nearest Neighbor Pattern when Interprocess Interference is Considered.

Figure 4.10 shows the effect of interprocess interference for the polling pattern. There is no noticeable change in the polling pattern when short messages are used. A slight increase is noticed for long messages when there is interference from another process. However, the amount of increase is relatively small and is negligible. In some cases, a small reduction in the message latency can be observed. This is due to the fact that the traffic spreads over a larger region and the contention on communication links is reduced. The traffic generated by these processors communicating with one another is spreaded over the allocated area. For communication intensive jobs, the communication latency benefits from this spreading if other jobs do not introduce much interference. Furthermore, a spread out area may allow more alternatives for routing paths while employing an adaptive routing scheme [63, 64]. The number of active packets in the network for polling pattern is small because the central node sends out its polling packets to its destinations sequentially. Similar observation can be derived for the random pattern with inference as shown in Figure 4.11.



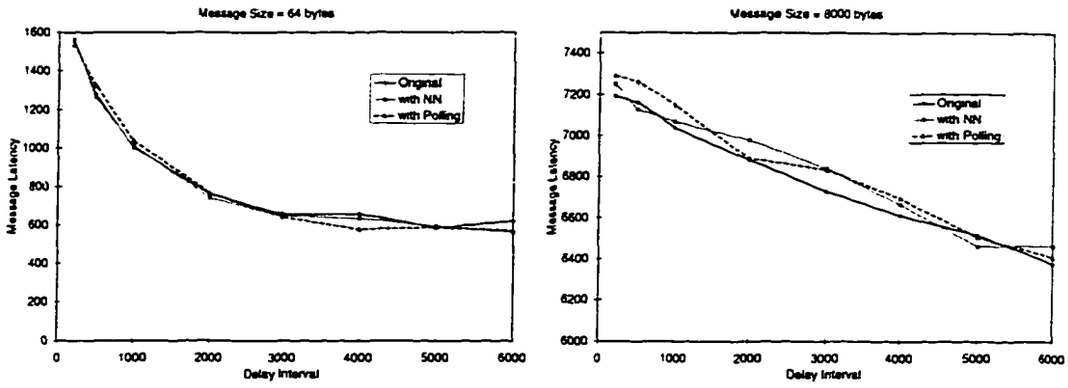
(a) (b)

Figure 4.10 Message Latency in the Polling Pattern when Interprocess Interference is Considered. (NN: Nearest Neighbor).

4.3.4 Effect on Job Execution Time

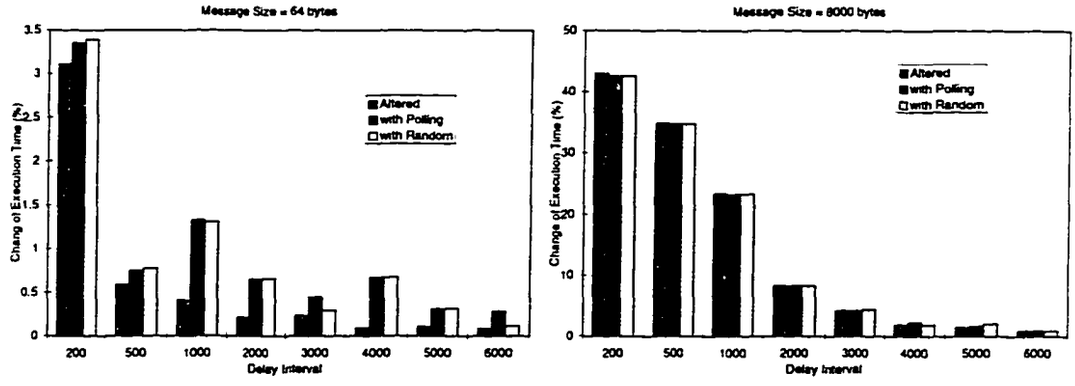
The execution time amounts to the total time a job spends on computations and communication. We have ignored the I/O operation delays. The change in execution time for nearest neighbor pattern is shown in Figure 4.12. Jobs requiring nearest neighbor communication pattern experience longer execution times when the geometry of nodes is altered. This is because of the longer message latency caused by the altered geometry. Interference from the polling and random patterns causes even higher change in the execution time of a job as shown in Figure 4.12. However, the increase diminishes for high delay intervals.

The effect of job interference for polling and random patterns are shown in Figures 4.13 and 4.14, respectively. Both negative and positive changes are observed for these two patterns. The changes are small (between -0.5% and 0.5% for the polling pattern, and -4% and 6% for the random pattern) for the short messages. It is more insignificant when the long messages are considered. The negative change is caused by the reduction of message latency as discussed in the previous subsection. However, delay caused by unbalanced synchronization negates this effect and increases the job execution



(a) (b)

Figure 4.11 Message Latency in the Random Pattern when Interprocess Interference is Considered. (NN: Nearest Neighbor).



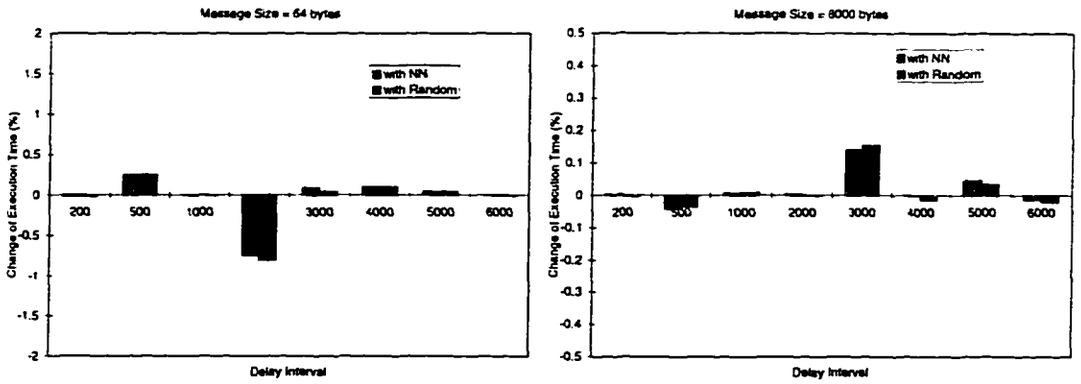
(a) (b)

Figure 4.12 Change of Execution Time for the Nearest Neighbor Pattern.

time in most cases. Nevertheless, interprocess interference have a smaller effect on job execution time compared to the effect of altered geometry.

4.4 Discussions

This chapter focuses on the effect of processor allocation on the message latency in a multicomputer system. The novelty of this work is that we performed actual measurements of message latency on a real system and have studied a few issues that were

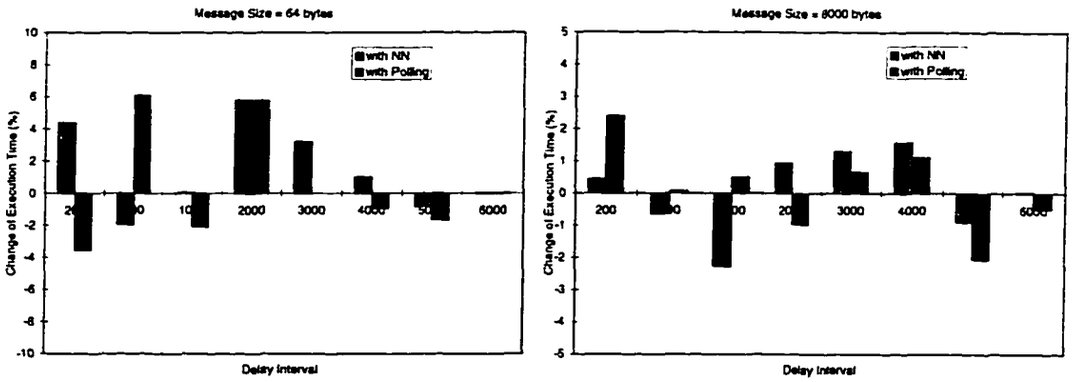


(a) (b)

Figure 4.13 Change of Execution Time for the Polling Pattern. (NN: Nearest Neighbor)

not analyzed earlier. We have also considered realistic traffic patterns along with synchronization constraints. Three communication models including both tight and loose synchronizations are implemented in our experiment. Message latency is measured for different size jobs and communication demands. Two issues in processor allocation, the geometry of allocated processors and the interprocess interference are evaluated. Both processor geometry and interprocess interference affect the communication paths and thus have direct impacts on the message latency.

Our results indicate a significant increase on the message latency if the geometry of the allocated processors is violated. This is caused by the alteration of the optimized communication paths from the original geometry. Parallel applications are usually developed to optimize the communication paths. When the optimized communication path is altered, contention for the communication links among messages causes the communication latency to increase. The worst case increase measured is as high as 45% if a long message size is used and the communication demand is high. Our results suggest to maintain the geometry of allocated processors to preserve the optimized communication paths. Although the results of interprocess interference indicate little or no impact on processes with low communication demands, there is still a possibility of increase in



(a) (b)

Figure 4.14 Change of Execution Time for the Random Pattern. (NN: Nearest Neighbor)

message latency in a large system or with a more scattered allocation. Partial contiguity among allocated processors prevents the interconnection network from being saturated and has to be maintained when possible. The multicomputer systems can benefit from a non-contiguous allocation scheme if the geometry and contiguity of the allocated processors can be retained to some extent.

5 ADAPTIVE NON-CONTIGUOUS ALLOCATION (ANCA)

5.1 Introduction

Processor allocation plays an important role in utilizing the processors for executing diverse applications. Conventional allocation algorithms [7]–[16] allocate a job to a set of contiguous processing nodes to minimize the distance of interprocessor communication path and to avoid the interprocess interference. A few recent effort have been focused on the non-contiguous allocations [25]. Both approaches have their pros and cons. In this chapter, we propose a novel allocation scheme, which in addition to contiguous allocation, adaptively allocates jobs to non-contiguous nodes. It combines the advantages of both approaches while avoiding the performance bottlenecks of them.

Performance of the contiguous allocation algorithms is limited by the fragmentation problem. Fragmentation occurs when there are free nodes in the system but the allocation algorithm fails to allocate these nodes to the waiting jobs. Significant performance improvement cannot be obtained by refinement of contiguous allocation algorithms because of the fragmentation problem associated with the nature of contiguous allocation [29, 32, 33, 34, 35]. Current switching techniques such as wormhole routing have made the communication latency less sensitive to the distance between communicating nodes and therefore makes non-contiguous allocation plausible. Liu et al. [25] have proposed several non-contiguous allocation strategies. However, their schemes are evaluated in a restrictive sense because of the underestimation of the increased communication latency.

The communication latency of a job allocated non-contiguously is increased because of the increased distance of the communication path and the contention of messages. Another factor that affects the communication latency is the geometry of the allocated nodes. Parallel applications and algorithms are optimized to minimize the number of communication steps and the distance of communication path [54]. Programs running on a mesh are developed to suit the mesh topology. Traffic patterns and communication paths are taken into consideration in the development of these programs. If the geometry of the nodes for an application is altered, the communication latency is expected to increase because the communication pattern of the application no longer remains optimized. Previously proposed non-contiguous allocations have not considered the geometry of the allocated nodes. Studies on wormhole routing techniques have shown that an unbalanced traffic pattern results in high message latency and can cause the intercommunication networks to saturate quickly [56, 57]. Thus a job allocated on an irregularly shaped submesh may create an unbalanced traffic pattern and can cause a high communication overhead. This effect is more serious considering the possibility of saturating the network. We therefore propose a partially non-contiguous allocation algorithm that solves the above problems.

The proposed scheme allocates a job to the required submesh size if available. Based on the availability of the processing nodes, a job may be broken into smaller subframes for allocation. All subframes of a job have to be allocated simultaneously to avoid synchronization problem. The number of times a job is divided is restricted by the proposed algorithm. The geometry of neighboring nodes is retained in the subframes and hence the communication overhead caused by the violation of geometry is avoided. The proposed scheme adaptively allocates jobs to non-contiguous submeshes and is therefore called adaptive non-contiguous allocation *(ANCA)* policy. The *ANCA* method improves performance by efficiently reducing the external fragmentation.

Because of the rapid advancing technology and diversity of applications, the cost

of communication latency is hard to estimate. Evaluation of the non-continuous allocation algorithms using approximate communication latency could be inappropriate and misleading. Therefore, we choose to simulate the ANCA policy under two extreme cases. An optimistic scenario indicates the best performance that can be achieved by the ANCA policy. A pessimistic (if not the worst) case is also simulated to study the limit on the potential gain of ANCA. The actual performance of the ANCA scheme can then be predicted by the results from these two scenarios. ANCA policy is compared with the first-fit algorithm under the two extreme cases. It is observed that a significant performance gain can be obtained with the proposed algorithm when the communication overhead caused by non-contiguous allocation is negligible. The study for the pessimistic scenario shows the importance of choosing an appropriate adaptability. High adaptability, although provides a better performance gain in the optimistic scenario, has the potential of saturating the network and thus does not guarantee a reasonable performance when communication overhead is considered. With low adaptability, the effect of extra overhead caused by non-contiguous allocation can be limited to a certain degree and thus guarantee the performance gain of the ANCA scheme. The probability of a job being allocated non-contiguously is also studied for ANCA schemes.

5.2 Adaptive Non-Contiguous Allocation *(ANCA)*

Contiguous allocation schemes are prone to physical fragmentation. To alleviate this problem, non-contiguous allocation which allows jobs to be allocated on scattered nodes can be implemented. Non-contiguous allocation has the potential of improving the system performance by reducing fragmentation. We propose an *adaptive non-contiguous allocation (ANCA)* scheme to improve the performance of mesh-connected multicomputers.

Non-contiguous allocation schemes minimize the queuing delay of jobs by allocating

the waiting jobs to non-contiguous processors. By executing jobs on non-contiguous processors, the communication latency is expected to increase. From Equation 1.1, the following observation can be made. Let ΔT_{queue} and ΔT_{comm} represent the time difference between contiguous and non-contiguous allocations. The idea of developing a non-contiguous allocation algorithm is to maximize the value of $\Delta T_{\text{queue}} - \Delta T_{\text{comm}}$ so that the turnaround time of a job in Equation 1.1 can benefit from shorter queuing delay without being penalized for the communication latency. Experimental study indicates that the queuing delay can be greatly reduced resulting in a high ΔT_{queue} if the fragmentation can be reduced [29, 42, 40, 32]. Design of faster switching devices and wormhole routing techniques have made the message passing latency insensitive to the communication distance making ΔT_{comm} negligible. Non-contiguous allocation is hence a very attractive alternative for processor allocation.

Non-contiguous allocations can be classified into two classes, totally non-contiguous and partially non-contiguous. In a totally non-contiguous allocation scheme, a job can be allocated as long as the number of available processors is sufficient for its execution. In a partially non-contiguous allocation, the nodes allocated to a job retain a certain degree of contiguity. During the execution of a process, processors assigned to a job communicate with one another. When contiguous allocation is used, the traffic is localized within the allocated submesh and causes no interprocess interference. With non-contiguous allocation, a message may have to travel through intermediate nodes assigned to other processes. Messages from different processes compete for the communication links. Messages that are blocked will have to be buffered and suffer extra overhead. Interprocess interference is expected to be less in a partially non-contiguous allocation because the local traffic in each contiguous region does not go through nodes assigned to other jobs and is less likely to collide with messages generated by other processes. Local communications within each region also cause no interference with messages generated by other processes except those messages that trespass through its territory. The study on the

effect of scattered processor allocation in a wormhole routed mesh [58, 59] agrees with this observation.

The message passing contention experiment in [25] shows that the number of communicating nodes is an important factor affecting the contention. The authors state that with small packet size and less than 9 pairs of communicating nodes, the effect of message contention is virtually negligible and hence support their non-contiguous allocation schemes. The effect of contention is more visible when the interconnection network is operated at a higher load. With diverse parallel applications, assumptions of small packet size and less intercommunicating nodes may not be valid. Another problem associated with message contention is its potential of causing network saturation. Once a network is saturated, the message latency grows infinitely large and greatly degrades the system performance.

Another issue which affects the turnaround time of a job is the geometry of the processors assigned to it. Parallel applications and algorithms are optimized to minimize the number of communication steps and the distance of communication path. If the relative locations of the nodes for an application is violated, the communication latency is expected to increase because the communication pattern of the application is no longer optimized. Previously proposed non-contiguous allocations do not consider the geometry of the nodes and will cause extra communication overhead. This overhead can degrade the performance of the system to a great extent. Therefore, a non-contiguous allocation algorithm which carefully considers the increase of communication latency has to be derived.

Based on these observations, we derive the following conclusions regarding a good non-contiguous processor allocation scheme.

- Non-contiguous allocation should not be applied when contiguous allocation is possible to avoid the increase in T_{comm} and the possibility of saturating the in-

terconnection network.

- When fragmentation prevents a job from being allocated, non-contiguous allocation can be applied to reduce T_{queue}.
- If a job has to be allocated non-contiguously, it is desirable to maintain some degree of contiguity to avoid increase of T_{comm} caused by interprocess interference.
- The geometry of nodes allocated to a job has to be retained as much as possible to avoid the extra communication overhead by disrupting the optimal communication pattern.
- For a job with high communication demand or system with slow communication network, contiguous allocation prevents the introduction of communication overhead and is preferred. System manager or the user should be provided with the flexibility of choosing the degree of non-contiguity.

The *adaptive non-contiguous allocation (ANCA)* scheme is proposed to meet the above requirements of a good non-contiguous allocation scheme. It always attempts to allocate a job contiguously. When contiguous allocation is not possible, it breaks a job request into equal-sized subframes. These subframes are then allocated to available locations and thus take advantage of non-contiguous allocation. Within each subframes, the relative positions among neighboring nodes is retained.

ANCA differs from existing allocation algorithms in two aspects. First, it combines the advantages of both contiguous and non-contiguous allocation schemes. Unlike other non-contiguous allocation algorithms which allocate all jobs non-contiguously, ANCA only allocates jobs non-contiguously when physical fragmentation occurs. Second, it preserves the geometry of nodes in jobs which is important but not considered in previous non-contiguous allocations.

5.2.1 ANCA Scheme

Before we present the ANCA scheme, some terminologies need to be defined for the ease of explanation. A mesh system is denoted by *M(w,h)* where w is the number of columns and h is the number of rows of processors. A job J is denoted by *J(m,n)* where m and n represent the number of columns and rows of processors, respectively, required for the execution of job J. A submesh can be identified by its lower-left corner which is referred to as its base.

Definition *Busy Array:* For a mesh $M(w,h)$, its busy array, B is an array in which the element $B[i,j]$ has a value 1 (0) if processor $< i,j >$ is busy (idle).

Definition *Coverage:* The coverage of an allocated submesh, I, with respect to an incoming task J is a collection of processors each of which, when served as the base of J will cause overlap between I and J. The union of the coverage of all the allocated submeshes is the coverage set of the system for the incoming task.

Coverage can be classified as left coverage and bottom coverage. Left coverage is the region of the nodes on the left side of an allocated job which cannot be served as the base of the incoming task. Bottom coverage is the region of the nodes below the left coverage and the allocated task which cannot be served as the base for the incoming job.

Definition *Reject Set:* The reject set with respect to an incoming task is the set of processors which can never serve as the base of an available submesh for accommodating the incoming tasks. The reject set contains the processors in the top rows and the right columns. An example of the coverage and the reject set for an allocated job is illustrated in Figure 5.1 in which the processors are represented by the intersections of the horizontal and vertical lines.

Definition *Coverage Array:* For a mesh $M(w,h)$, its coverage array with respect to an incoming job $J = (w', h')$ is $C[w - w' + 1, h - h' + 1]$, in which $C[i,j]$ is equal to 1 (0) if

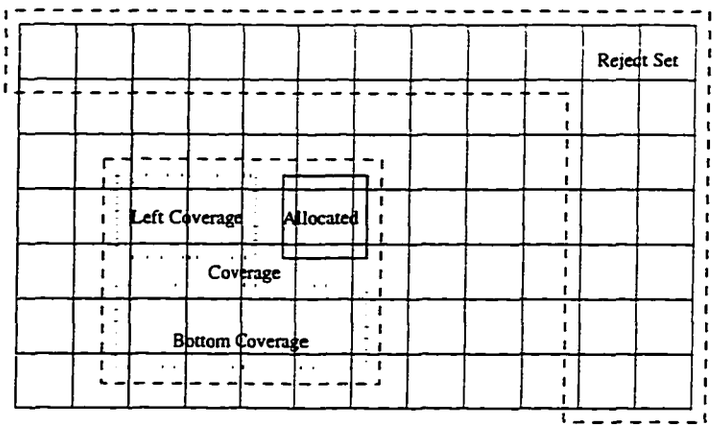


Figure 5.1 Coverage and Reject Set with Respect to a Submesh Request of Size $< 4, 3 >$.

processor $< i, j >$ is (is not) in the coverage set. A 0 in the coverage array indicates the location for an available submesh for accommodating the incoming job. The coverage array has a size of [w-w'+1, h-h'+1] which is smaller than the mesh size. This is because the processors in the reject set cannot be served as the base for the incoming task and are not included in the coverage array.

We define *adaptability* as the measure of the allocator's ability to adjust its allocation policy according to the system status. It is quantified by the number of times a job request is splitted into smaller subframes. An adaptability of 0 refers to a strictly contiguous allocation. System administrators are allowed to decide the maximum adaptability for their system's needs. Individual applications can also specify their adaptability to allow communication intensive jobs to be allocated contiguously to minimize the interconnection latency.

The ANCA scheme consists of two parts, a decomposition process and an allocation process. Decomposition process splits a job into smaller subframes for allocation when fragmentation prevents a job from being allocated to a required submesh size. The allocation process is used to check for the available processors. An ANCA scheme that allows splitting of jobs for a maximum of A times (i.e. with maximum adaptability

of jobs set to A) is denoted as ANCA-A. ANCA always attempts to allocate a contiguous submesh to a job request whenever possible. If contiguous allocation fails, the decomposition process is used. A new subframe size is generated for allocation in every attempt. The allocation algorithm is then used to locate the number of subframes which can satisfy the requirements of the job. When enough subframes are found, the job is allocated. Otherwise, the decomposition and allocation process is repeated until the job is allocated or the maximum adaptability is reached. We begin our discussion of the ANCA algorithm with the decomposition and the allocation processes followed by the complete algorithm.

Decomposition Process: The decomposition process splits the current subframes into smaller subframes. The generated subframe size for the next allocation attempt equals to one half of the current subframe size. There are three reasons for using a subframe as the allocation unit. First, we want to maintain a certain degree of contiguity in every allocated region. By splitting a subframe into half of its current size, the sizes of all subframes are equal except those on the edges. Therefore, the subframes of a job maintain similar contiguity. Second, the geometry of nodes can be easily preserved using the subframes with a simple direct mapping mechanism. Third and the most important reason for using a subframe as the allocation unit is to simplify the complexity of the allocation process. Our allocation algorithm uses a bit-array approach as used by most of the contiguous allocations. Scanning the bit-arrays for possible allocation is the most time-consuming part in the allocation algorithm. By using subframe as an allocation unit, all free subframes in the system can be located by scanning the bit-array in only one pass.

In some cases, the resulting subframe size is not an integer and has to be rounded up to the nearest integer. Internal fragmentation is introduced due to the rounding up effect. This problem is solved by not allocating excessive nodes. A bookkeeping mechanism is used to keep track of the exact number of processors needed for a job and therefore avoids

allocating excessive nodes to a job. Two counters are used is the bookkeeping, $x_{\text{allocated}}$ and $y_{\text{allocated}}$. A row major order is used to allocate the subframes. Upon allocation of a subframe, a direct mapping of nodes from the original request to the subframe is performed. After a subframe is allocated, the value of $x_{\text{allocated}}$ is incremented by its size in the x dimension. If $x_{\text{allocated}}$ becomes larger than the job size in the x dimension, $x_{\text{allocated}}$ is reset to zero and $y_{\text{allocated}}$ is incremented by the subframe size in the y dimension. When $y_{\text{allocated}}$ becomes larger than or equal to the job size in the y dimension, we would have the required number of subframes allocated. Figure 5.2 shows how a 5 × 2 submesh is splitted into subframes with adaptability 1 and 2. It also demonstrates the mapping of the processors from the original request to the subframes and how excessive nodes are discarded for actual allocation.

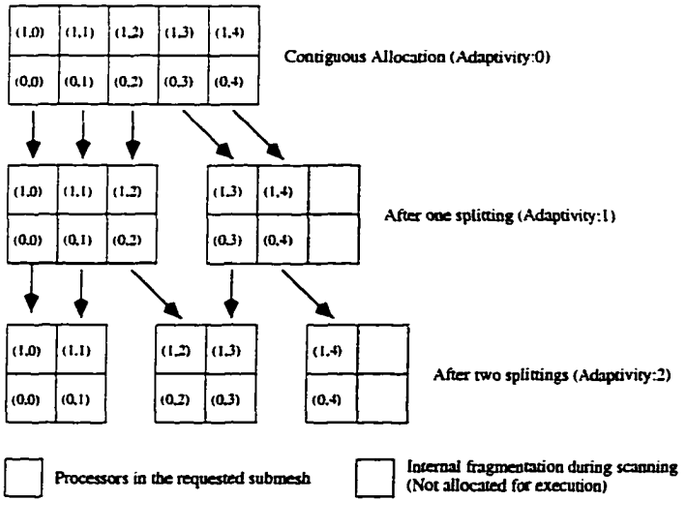


Figure 5.2 Example of the Decomposition Process.

Allocation Process: The first-fit algorithm [9] is extended for ANCA. The main consideration is to allow the co-allocation of multiple subframes with minimal complexity. A coverage array is generated for the allocation of subframes. It is then scanned in a row-major order to locate the free subframes as candidate subframes for the allocation. The base of a candidate subframe is labeled as a candidate. When a candidate is found,

the candidate subframe and its left coverage have to be marked as unavailable to prevent the allocation of other subframe that overlaps with it. The bottom coverage does not need to be marked because the scanning sequence of the rows prohibits the algorithm from locating a candidate in another candidate's bottom coverage. Thus the time required for marking the coverage array for bottom coverage is saved. If ANCA fails to allocate a job using a particular subframe size, it can either split the subframe and retry with a smaller subframe size or enqueue the job if the maximum adaptability is reached.

Complete Algorithm: The complete algorithm for *ANCA-A* is presented in Figure 5.3. First, the coverage array is generated for the current subframe size. After the coverage array is generated, the ANCA algorithm scans all the rows in the coverage array from bottom up. In every row, the elements are scanned from left to right. If a 1 is found at C[i,j], node $< i, j >$ is labeled as a candidate for the allocation. The candidate subframe and its left coverage is then marked as unavailable to avoid the allocation of other subframe that overlaps with the candidate found for allocation. The counters, $x_{\text{allocated}}$ and $y_{\text{allocated}}$ are updated according to the bookkeeping process described earlier. The scanning for the 1's in the C array continues until $y_{\text{allocated}}$ becomes larger than or equal to the required submesh size in the y dimension. If the scanning fails to find the required number of subframes for a job after exhausting the C array and the limit on the splitting is yet to be reached, the job is further decomposed and retried for allocation.

The processor relinquishment is simple. When a job departs, the busy array is updated according to the nodes released by the departing process. The system is then signaled for allocation of jobs waiting in the queue.

5.2.2 Complexity Analysis of the ANCA Algorithm

The space complexity of the ANCA algorithm involves storage of the busy array, the coverage array, and the temporary storage for the candidate subframes. It is same

Step 1 Let (w,h) be the system size, J=(m,n) be the job to be allocated. Let (w',h') be the size of the subframe for allocation attempt. Set w'=m, h'=n. Set adaptability of job, a, to 0.

Step 2 Fill the allocated submesh and their left coverage in the C array with the following procedure. Scan all the rows in B[w,h]. Scan each row from right to left and initialize left.cov:=w+1. Check B[i,j], if B[i,j]=1, set left.cov:=max(i-w'+1, 0). If $i \geq left.cov$, *then set C[i,j]=1, else set C[i,j]=0.*

Step 3 Fill the bottom coverage from C generated in step 2. Scan all columns in C from left to right. Scan each column from top to bottom and initialize btm.cov:=h+1. Check element C[i,j], if C[i,j]=1 set btm.cov:=max(j-h'+1,0). If $j \geq btm.cov$, *set C[i,j]=1.*

Step 4 Initialize the counters, $x_{allocated}$ *and* $y_{allocated}$ *to 0.*

Step 5 Scan all rows in the C array from bottom up. Scan elements in each row from left to right. At element C[i,j], if C[i,j]=1, do nothing. Else,

(a) Set $< i, j >$ *as a candidate. Mark the candidate subframe at* $< i, j >$ *and its left coverage as unavailable in the C array.*

(b) Increment $x_{allocated}$ *by* w'. *If* $x_{allocated} \geq w'$, *increment* $y_{allocated}$ *by* h' *and reset* $x_{allocated}$ *to 0.*

(d) If $y_{allocated} \geq y'$, *enough candidates are found, goto step 7, else continue the scanning from* $< i, j >$.

Step 6 If $a = A$, *goto step 7. Else, decompose subframe (w',h') into smaller subframes. Increase a by 1. Set (w', h') according to the size of the new subframes. If both w' or h' becomes 1 after the decomposition, goto step 8, else goto Step 2.*

Step 7 Allocate the job to the candidates found in step 5. To allocate the next job in the queue, goto step 1.

Step 8 Allocation failed. Put the job in the system queue and wait for the departure of an executing job for retrial.

Figure 5.3 ANCA-A Algorithm.

as the first-fit algorithm except that in ANCA additional storage is required for the candidates. The time complexity for ANCA in one iteration is slightly worse than the first-fit algorithm because of the extra time taken for the marking of the candidate subframes. The marking of the candidates in one iteration takes $O(mn)$. For allocation of an $m \times n$ job in a $w \times h$ mesh, the big O representation for the time complexity is $O(wh + mn) = O(wh)$. Therefore the time complexity of the ANCA-A algorithm to allocate a job is equal to $O(Awh + Amn)$.

5.3 Performance of the ANCA Scheme

5.3.1 Simulation Model

The ANCA scheme is studied with event-driven simulations. The simulated system is a 32×32 mesh. The system parameters such as the arrival process of the jobs and the distribution of job sizes are the same as in Table 3.1 and 3.2. Previous chapter studies the effect of processor allocation to the communication latency of parallel jobs in multicomputer systems. The results in that chapter indicate the important correlation between processor allocation and communication but no solid latency model is derived to show the amount of communication latency affected by the processor allocation schemes. Due to the lack of a latency model to estimate the increased communication latency in a non-contiguously allocated environment, two different scenarios are simulated for the ANCA scheme.

An optimistic scenario assuming no communication overhead is simulated. It indicates the best performance that can be achieved by using ANCA. In the simulation, the execution time of a job remains unchanged regardless of whether it is allocated non-contiguously or not. This scenario reflects the possible performance of systems with efficient interconnection network or jobs with low communication demand.

Communication overhead is exaggerated in the pessimistic scenario to estimate the

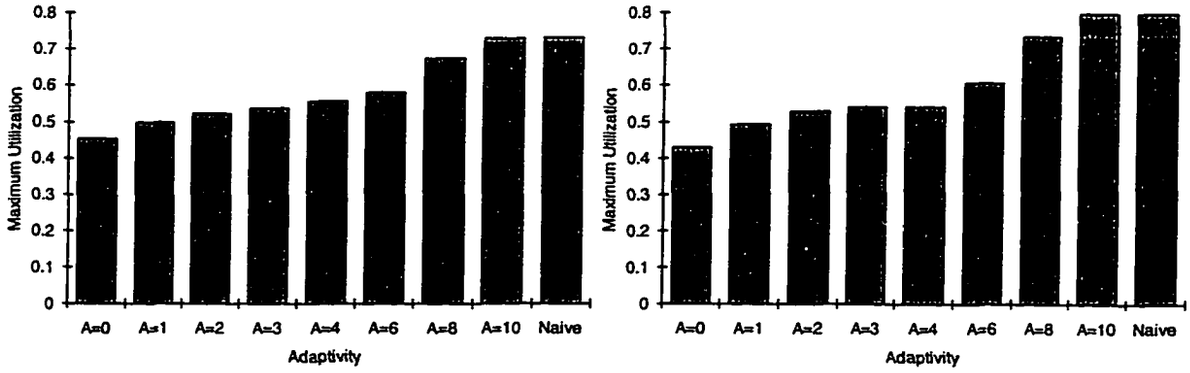
limitation of ANCA policy in an operational mesh. Previous study [56] indicates an important characteristic for the message delay of wormhole routed meshes. In a wormhole routed mesh, message delay are relatively constant to various arrival rate of messages. It grows slowly when the arrival rate of a message increases. Higher message arrival rate means more contention for the communication links in the network. Once the message arrival rate increases beyond the system's capacity, the network saturates and the message delay goes out of bound. Typically the effect of message contention increases the average packet delay by less than three folds. We devise our pessimistic scenario based on this characteristic. Another factor involved in the communication overhead is the ratio of communication time over the total execution time for a job allocated non-contiguously. The ratio of communication time to computation time for a job in multicomputers is usually low. Our pessimistic scenario is devised to predict the limitation of ANCA policy. It attempts to imitate the behavior of a near-saturation mesh. Therefore, we penalize every non-contiguously allocated job by increasing its communication latency by three times. This is higher than the latency incurred in most networks. We also set the communication time of a job at a high value of 30% of its total execution time. This is done to ensure that most application will have a lower communication demand. By exaggerating the message transfer delay and the communication demand of jobs, the pessimistic scenario reflects the behavior of a near-saturation mesh with communication-intensive jobs. In the pessimistic simulation, when a job is allocated non-contiguously, its communication time is increased by three times to reflect the communication overhead caused by non-contiguous allocation. The actual performance of a particular ANCA algorithm can be predicted with the two curves drawn for its turnaround time in the optimistic and pessimistic scenarios. However, this prediction is only valid when the communication network is not saturated. Therefore, we also discuss the percentage of jobs being allocated contiguously which is directly related to the possibility of network saturation.

5.3.2 Optimistic Scenario

Figure 5.4 illustrates the maximum processor utilization of ANCA policy allowing different adaptability at a very high traffic ratio. Adaptability zero is equivalent to the first-fit algorithm. Naive allocation [25] is also included in this comparison. Naive allocation allocates processors to the incoming jobs in a predefined scanning order. A job can be allocated as long as the number of processors in the system is sufficient for its execution. It is free of any fragmentation and indicates the best utilization obtainable. Other non-contiguous allocations proposed by Liu, etc. [25] are also free of fragmentation and have the same maximum utilization as the naive allocation. The first-fit algorithm has the worst utilization because of the external fragmentation of second kind as defined in Section 2.1.4. ANCA allows jobs to be allocated on small subframes and therefore reduces fragmentation. By allowing a higher adaptability, ANCA provides a better utilization of the processors in the system. This is true with both job size distributions – uniform and normal – considered in our simulations. With adaptability 10, jobs can be splitted into subframes with only one processor and is thus free of fragmentation. It has utilization equal to the naive algorithm.

In most cases, utilization provides a measure of the system performance. However, in some cases, it can be misleading. As discussed earlier, partial contiguity and geometry of nodes are important to lower the communication overheads. Therefore, the high utilization achieved by non-contiguous allocation may not mean better performance. At the same utilization, ANCA will provide better performance than other non-contiguous allocation algorithms because of the partial contiguity and geometry it provides.

The most important metrics in evaluating an allocation policy is the average turnaround time of a job. The turnaround time includes both queuing delay and execution time of a job. Figure 5.5 illustrates the average turnaround time of a job with different adaptability. Both uniform and normal distribution of job sizes show similar trend. As expected,



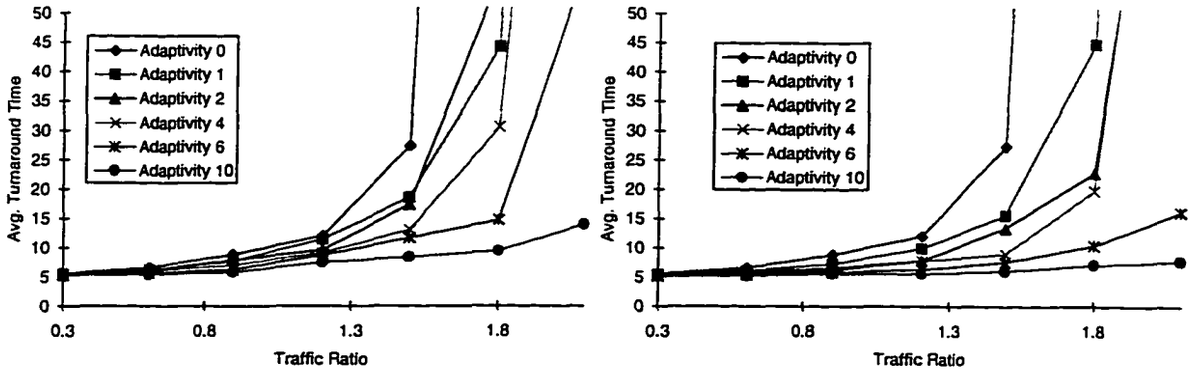
(a) Uniform Sized Jobs (b) Normal Sized Jobs

Figure 5.4 Comparison of Maximum Utilization Allowing Different Adaptability.

higher adaptability provides lower turnaround time because of the reduction of fragmentation. This is a result of the reduction of the external fragmentation of the second kind. The difference is noticeable with a small increase of adaptability from 0 to 1 or 2. The improvement on average turnaround time is especially significant at traffic ratio higher than 0.9. For lower traffic, queuing is seldom encountered and the turnaround time is mainly dependent on the execution time of the jobs. At higher traffic, queuing delay becomes a dominating factor in the average turnaround time. By applying ANCA, fragmentation is reduced with the increase of adaptability and therefore the queuing delay reduces. Without using ANCA, the system gets saturated quickly after the traffic ratio gets higher than 1.2. ANCA allows the system to work at higher traffic without saturating.

5.3.3 Pessimistic Scenario

Figure 5.6 illustrates the average turnaround time for different adaptability when the pessimistic scenario is assumed. Contrary to the observation of the optimistic scenario, the average turnaround time does not improve with higher adaptability. ANCA-1 algorithms performs slightly better than the first-fit algorithm with uniform job size

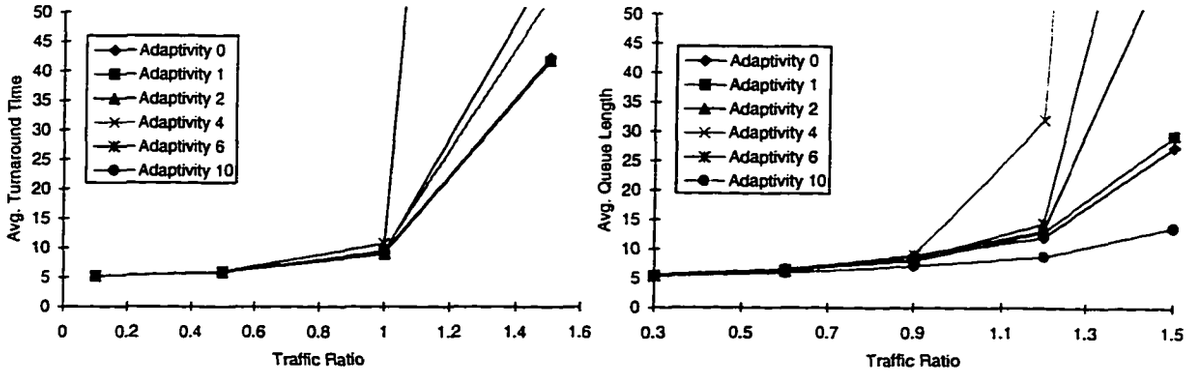


(a) Uniform Sized Jobs (b) Normal Sized Jobs

Figure 5.5 Average Turnaround Time for ANCA with Different Adaptability.

distribution under low traffic. In all other cases, ANCA performs worse than the first-fit algorithm except ANCA-10 in the normal job size distribution case. ANCA-4 and ANCA-6 have the worst performance among all policies tested. The poor performance is a result of exaggerating the communication overhead. Higher adaptability allows larger jobs to be allocated with more splitting and is less likely to be affected by fragmentation. It also tends to allocate more jobs non-contiguously. Execution time of jobs allocated non-contiguously are assumed to increase and therefore higher adaptability results in higher turnaround time. With the increase of adaptability, fragmentation is also reduced. In the case of ANCA-10, the communication overhead is offset by the reduced queuing delay and therefore produces lower average turnaround time compared to ANCA-4 and ANCA-6. For normal job size distribution, job sizes tends to concentrate at half of the system size in each dimension. Less fragmentation is expected because of the concentration of job sizes. Therefore, with ANCA-10, the fragmentation can be reduced by a great extent and a turnaround time lower than ANCA-0 is observed.

Another study conducted in this pessimistic scenario is the possibility of saturating the interconnection network. As discussed in Section 5.2, non-contiguous allocation may cause the interconnection network to saturate easily and therefore result in an un-

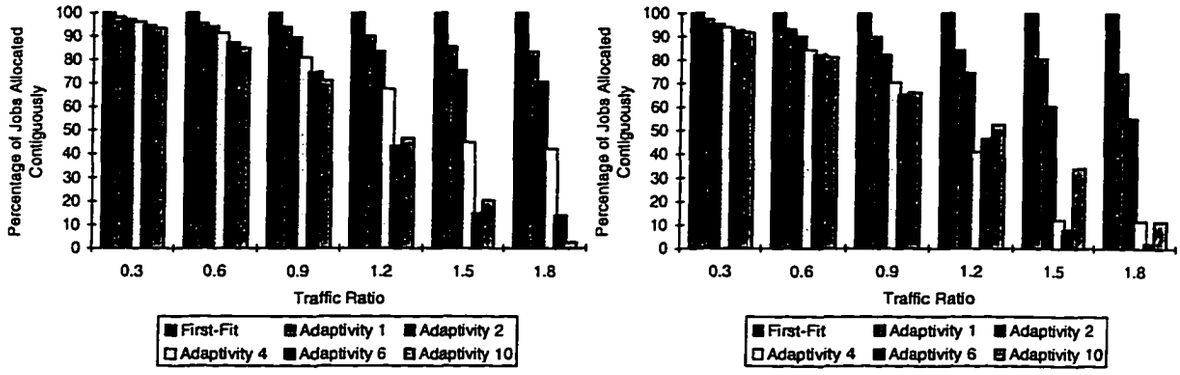


(a) Uniform Sized Jobs (b) Normal Sized Jobs

Figure 5.6 Average Turnaround Time for Different ANCA Policies in the Pessimistic Scenario.

operational system. We compare the percentage of jobs being allocated contiguously in Figure 5.7. Higher percentage of contiguous jobs reflects a lower communication overhead and can be interpreted as a lower possibility of saturating the interconnection network. The first-fit algorithm is a strictly contiguous allocation scheme and hence allocates all jobs contiguously. Percentage of contiguous jobs drops when traffic increases because fragmentation occurs more often and requires more jobs to be splitted. Higher adaptability have lower percentage of contiguous jobs under low traffic. At high traffic, medium adaptability such as ANCA-4 and ANCA-6 allocates less contiguous jobs because these policies have longer queuing delay and therefore splits more jobs. In the worst case, ANCA algorithm still allocates at least 2.3% of jobs contiguously. With a lower adaptability set for the system, we can ensure a high percentage of contiguous jobs and therefore prevent the interconnection network from saturating. For example, ANCA-1 allocates 83.5% and 74.6% of jobs contiguously in the uniform and normal distribution cases, respectively. Previously proposed non-contiguous allocation schemes do not have this ability to maintain a high percentage of contiguously allocated jobs and therefore have a potential of saturating the network and causes an unoperational system. For example, the MBS scheme provides partial contiguity and is believed to have the best

performance among Liu's algorithms [25]. In the MBS, a job can be allocated without changing its submesh request only when its size equals to a buddy and when a buddy of that size is available. In a 32 × 32 mesh, only square jobs with side lengths equal to 1, 2, 4, 8, 16, and 32 can be allocated contiguously. Assuming a uniform distribution, the probability of having such a submesh request is equal to 6/1024. This probability has to be multiplied with the probability of having an available buddy of the equal size and therefore the actual percentage of jobs can be allocated contiguously in the MBS scheme is much lower than 6/1024. It is very possible for the network to saturate with so many jobs allocated non-contiguously, and in addition, have their geometry altered.



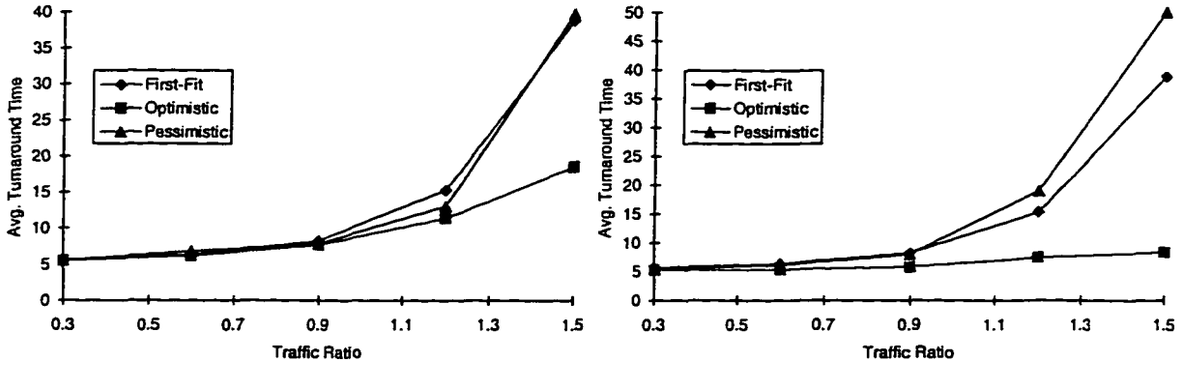
(a) Uniform Sized Jobs (b) Normal Sized Jobs

Figure 5.7 Percentage of Contiguously Allocated Jobs.

5.3.4 Performance Prediction of ANCA Scheme

Because of the rapid advancing technology and diverse applications, it is difficult to accurately predict the performance of a non-contiguous allocation. Instead of assuming a model for a specific workload as done in previous proposed non-contiguous allocations, we chose to show the potential performance of the ANCA scheme with the two scenarios simulated. The average turnaround time of a system implementing ANCA algorithm would lie in between the two curves: the optimistic case and the pessimistic case provided the network is not saturated.

Figure 5.8 illustrates a comparison of the expected performances of ANCA-1 and ANCA-10 compared to the first-fit algorithm for uniform job size distribution. ANCA-1 performs better than the first-fit algorithm in most cases. Only at traffic ratio of 1.5, ANCA-1 has higher turnaround time than the first-fit algorithm in the pessimistic case. The difference is insignificant compared to the turnaround time. Therefore, a better performance than the first-fit scheme can always be expected using ANCA-1. The curve drawn for the first-fit algorithm falls within the region bounded by the two curves drawn for ANCA-10. This indicates a possible performance degradation using ANCA-10 although the possible performance gain of ANCA-10 from the optimistic case is higher. ANCA with other adaptability exhibits the same trend as ANCA-10 and therefore ANCA-1 is the only scheme which can guarantee a performance improvement with the uniform job size distribution.



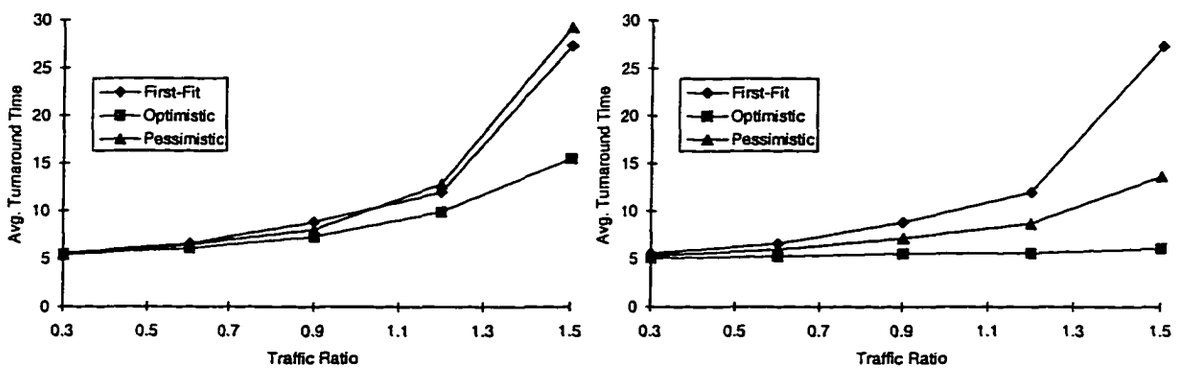
(a) ANCA-1 (b) ANCA-2

Figure 5.8 Predicting the Average Turnaround Time for ANCA.

The simulation result for the normal job size distribution is illustrated in Figure 5.9. The pessimistic case result for ANCA-1 is slightly worse than that of the first-fit. There is a small possibility of worse performance than the first-fit algorithm using ANCA-1. The results obtained for ANCA with other adaptability show similar trend except ANCA-10. Both of the optimistic and pessimistic scenarios provide better performance than the first-fit with ANCA-10. Better performance is expected provided that the network

is not saturated. However, as studied in the previous section, ANCA-10 allocates only a few jobs contiguously and has a high possibility of saturating the network. On the other hand, ANCA-1 allocates 74.5% of jobs contiguously and introduces less communication overhead. Low adaptability is expected to provide performance improvement without saturating the network.

In our results, ANCA-1 provides good performance improvement in the uniform job size distribution case. It also provides reasonable (if not better) performance when normal distribution is assumed for job size. The curves for both the optimistic and pessimistic scenarios for ANCA scheme can be refined to provide a better prediction of its performance with better knowledge about the speed of the communication network and the characteristics of the application programs. Because the communication overhead for the near-saturation network is exaggerated in our simulation, the average turnaround time for the pessimistic scenario may be lower. Therefore, ANCA with higher adaptability may also be suitable for implementation if the communication overhead can be limited.



(a) ANCA-1 (b) ANCA-10

Figure 5.9 Predicting the Average Turnaround Time for ANCA.

5.4 Discussion

The ANCA allocates the requested submeshes if possible and hence minimizes the communication latency. When fragmentation prevents a job from being allocated, it splits the request into smaller subframes and allocates the job to the subframes. Each subframe maintains a certain degree of contiguity and geometry. The proposed scheme allows system administrator to choose an optimal value for adaptability. Adaptability can also be specified for individual jobs to prevent a communication intensive job from being allocated non-contiguously.

Simulation study indicates a significant performance gain when the communication overhead caused by non-contiguous allocation is negligible. Jobs that have less communication demand or systems with high speed interconnection network are most likely to benefit from the ANCA policy. The pessimistic scenario simulation indicates a potential limit on the performance gain of the ANCA scheme. High adaptability of system causes many jobs to be allocated non-contiguously while introducing more communication overhead. Possibility of ANCA with different adaptability is also discussed. Our results prefers ANCA-1 over other ANCA policies because of its performance improvement and low possibility of saturating the network.

6 AN INTEGRATED PROCESSOR MANAGEMENT SCHEME

Both processor allocation and job scheduling schemes have been proved to improve the performance of multicomputer systems. It is desirable to combine the advantages of both processes into an integrated processor management scheme. However, the complexities associated with several proposed allocation algorithms and job scheduling policies make the integration infeasible. In order to develop a feasible integrated processor management scheme, both the allocation algorithm and job scheduling policy have to be simple and efficient.

In this chapter, we propose a job scheduling strategy based on a simple *bypass-queue (BQ)* technique. The implementation of the BQ scheme does not require additional storage with respect to the first-come-first-serve queue. It also incurs a very low overhead to perform the job sequencing process. To take further advantage of the bypass-queue scheduling, a *fixed-orientation (FO)* allocation algorithm is also proposed. The proposed FO algorithm allocates jobs in a fixed orientation so that the physical fragmentation is reduced. It has lower complexity than the other allocation algorithms. An integrated processor management scheme can therefore be implemented by combining the BQ scheduling and the FO allocation because of their low complexities. This section describes the BQ scheduling and the FO allocation followed by the integrated processor management scheme.

The following sections discuss the proposed bypass-queue scheduling and the fixed-

orientation allocation algorithms. The performance evaluation of the proposed schemes are presented in the end of this chapter.

6.1 Bypass-Queue (BQ) Scheduling

Because of the disadvantages of multiprogramming in multicomputers, scheduling in the multicomputers has been focused on sequencing the order of execution for the arrived jobs. It is observed that multicomputers are often underutilized because of the blockade situation associated with the FCFS system. Many processors can be left idle even when there are jobs waiting. By executing the jobs in a carefully arranged order, the blocking effect of the FCFS queue can be diminished as proved in the schemes reported in [33, 34, 35, 38, 37]. However, these schemes require large storage space for implementation of the required multiple queues and are also complicated from an implementation standpoint.

We propose a bypass-queue policy to schedule job requests in the mesh system to solve the problems associated with the other scheduling strategies. A bypass queue is a variation of the FCFS queue without the blocking problem. When the queue is activated for allocation, the waiting jobs are checked for allocation in the order of their arrival as is done in the FCFS queue. However, a job is allowed to bypass the unallocated jobs ahead of it if it can be allocated. The process continues until all the jobs in the queue have been checked or an executing job departs from the system. In case of a departure, the entries in the queue are checked for allocation starting from the head of the queue. By allowing a job to bypass the unallocated jobs, the performance is benefited in two ways. First, the turnaround time of the jobs that get ahead are efficiently reduced. Second, the system is better utilized and is therefore able to provide better overall performance.

To ensure that every job can obtain the service after a reasonable waiting time, a threshold time is set for the system. The threshold time is defined as the maximum time

a job allows other jobs to bypass it before getting allocated. If any of the jobs waits longer than the threshold time, the bypassing is disabled and the jobs are only served in the FCFS manner. Because of the threshold time, a job only has to wait for the release of the occupied nodes by other jobs if it has waited longer than the threshold time and is at the head of the queue. The performance of this scheduling scheme varies with respect to the threshold time as discussed and quantified in Section 6.4.

The BQ scheduling scheme does not require multiple queues to store jobs and can be efficiently implemented with a linked-list. Jobs in the queue are represented as entities in the list according to the order they arrive. The allocation of a job bypassing the others is done by removing it from the list. The only extra operation compared to the simple FCFS system is the monitoring of the threshold time. The algorithmic simplicity and the minimum storage requirement makes the BQ scheduling extremely appealing.

6.2 Fixed-Orientation (FO) Allocation

Processor allocation algorithms can be classified as contiguous or non-contiguous. The feasibility of non-contiguous allocation algorithms [25] depends on the trade-offs between communication latency and waiting delay. In this chapter, we consider only the contiguous allocation scheme. Among the contiguous algorithms, the schemes proposed in [10, 11, 12] have drawn most attention because of their superior performance. The better performance of these algorithms is achieved by allocating jobs in alternative orientations. In adaptive-scan [10], the system is scanned for the available submesh requested by the jobs. When a requested submesh is not available, it is rotated by 90° and the system is checked for the rotated submesh. This increases the possibility of allocating a job into the system. The list-based algorithms [11, 12] also rotate a submesh for possible allocation. They maintain a list of the allocated [11] or free [12] submeshes and the allocation is done by checking the list instead of the bit-arrays commonly used by

other algorithms. The list-based algorithms claim to have lower time complexities than the adaptive-scan. However, the number of steps of operations for these algorithms do not directly reflect the time required to perform the allocation process. The list-based algorithms require complicated manipulation of a linked-list while the bit-array operations used in the adaptive-scan is relatively simple. The actual time taken to perform these algorithms is hard to compare because of the difference in the basic operations.

We propose a fixed-orientation algorithm for the mesh systems. Instead of changing the orientation of a job after the allocator fails to find a suitable mesh, we allocate all rectangular submesh requests in the same orientation. Either the width is always equal to or larger than the height or the height is always equal to or larger than the width. The orientation for allocation is chosen according to the orientation of the system. If a job requests a submesh with different orientation than the chosen orientation, it is rotated before allocation. Changing of the orientation requires translation of the logical addresses of the processors in the allocated submesh. This translation of logical address for processors is a trivial process and is described in [10].

The FO allocation has two advantages over the other algorithms. First, the allocation time is smaller compared to the adaptive-scan and list-based algorithms. Because all jobs are allocated in a fixed-orientation, our algorithm only needs to check for the available submesh in one orientation. The adaptive-scan and the list-based algorithms consider two different orientations for each submesh and hence may take twice the time to perform the allocation. Second, the fixed-orientation allocation has little virtual fragmentation. Consider the example shown in Figure 6.1 with jobs arriving in the labeled order. Because of the inefficient processor allocation, job 4 ends up being blocked as shown. By forcing all the jobs to be allocated in the same orientation, all 4 jobs can be accommodated. Also, it can be observed from this example that the location to place a submesh does affect future allocations. If job 3 is placed on the right of job 2 instead of on top of job 1, job 4 would be rejected for allocation. It is important to check the

possible allocation according to the orientation chosen. For a system which has more columns than rows, the x dimension has to be checked first, while for a system has more rows than columns, the y dimension needs to be checked first. This helps better utilize the space in the mesh and provides better possibility for future allocations. There is still a small possibility of virtual fragmentation due to the dynamic departure of the jobs. However, the simulation results in Section 6.4 indicate that the few virtual fragmentation associated with fixed-orientation algorithm has insignificant impact on its performance.

The implementation of the FO allocation requires associating a flag, *_rotated,* with each and every arriving job. Upon arrival, a job's orientation is checked. If it is different from the orientation used for allocation, the flag *_rotated* will be set and the job is rotated by 90^o. After the orientation is checked and the flag is set, a job is moved to the system queue for allocation. Jobs waiting in the queue are allocated according to the service discipline of the scheduling strategy. Any contiguous allocation algorithm can be used for locating the submesh in the chosen orientation. Upon allocation of a rotated job into the system, proper address translation of the processors can be performed as discussed in [10].

6.3 The Integrated Processor Management Scheme

To take advantage of both processor allocation and job scheduling, the proposed BQ scheduling and FO allocation are used in combination as an integrated processor management scheme. Their low complexities are the main reasons for the integration.

To describe the algorithm of our integrated processor management scheme, we first introduce the variables and data structure. A flag *_rotated* is defined as mentioned in Section 6.2 for every job in the system. It is set upon the arrival of a job and used to determine whether address translation is required when the job is allocated. A linked list is used to implement the bypass-queue. All the waiting jobs are appended to the end of

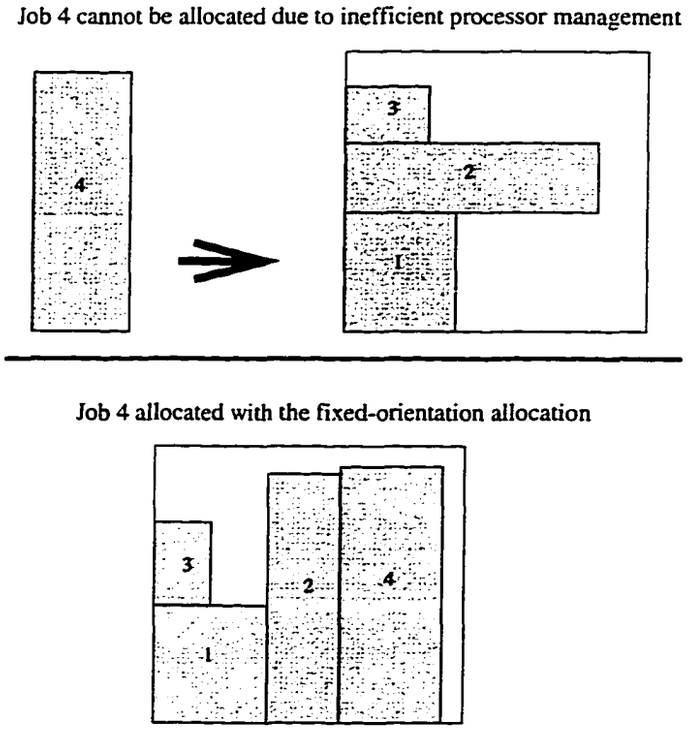


Figure 6.1 Reducing Virtual Fragmentation with the Fixed-Orientation Allocation.

the list for future allocation. The information contained in the linked list include the size, submission time, and the flag *_rotated* of the waiting jobs. Variable *threshold_time* is the threshold time set for the bypass-queue scheduling while *earliest_submission* represents the submission time of the job at the head of the queue.

The integrated scheme consists of two main processes, *job arrival* and *job departure*. A formal description of these processes is listed in Figure 6.2. The complexity for the FO allocation is equal to the first-fit and best-fit algorithm. It is lower than the complexity of the adaptive-scan and list-based algorithms because it does not have to check for the alternative orientation of the submesh. The manipulation of the bypass-queue does not introduce any significant overhead to the FCFS system. It does increase the number of allocation attempts because of the bypassing. Therefore, it is even more important to use an allocation algorithm with a low complexity such as the FO algorithm. The

value of threshold time affects the performance of the integrated scheme. A system administrator can determine the threshold time based on individual system's need. The performance evaluation of our processor management scheme is presented in the next section along with the discussion on choosing a proper threshold time.

Job Arrival:

Step 1 Check the orientation of the incoming job. Change the orientation and set _rotated *if necessary.*

Step 2 If queue is not empty, append the incoming job to the tail of the queue. Goto step 4.

Step 3 If queue is empty, check for allocation of the incoming job. If job is successfully allocated, assign processors to it and perform the required address translation when _rotated *is set. Otherwise, put the job at the head of the queue.*

Step 4 Wait for next arrival or departure.

Job Departure

Step 1 If queue is empty, goto step 4, else set earliest_submission *to the submission time of the first job in the queue. Choose the first job in queue as the candidate for allocation.*

Step 2 If candidate is allocable, assign nodes to the candidate with proper address translation (if required) and remove it from the queue. If the last job in the queue has been checked, goto step 4.

Step 3 If (current_time − earliest_submission < threshold_time), set the next job in queue as candidate. Set earliest_submission *to the submission time of the first job in the queue and goto step 2.*

Step 4 Wait for next arrival or departure.

Figure 6.2 The Integrated Processor Management Scheme.

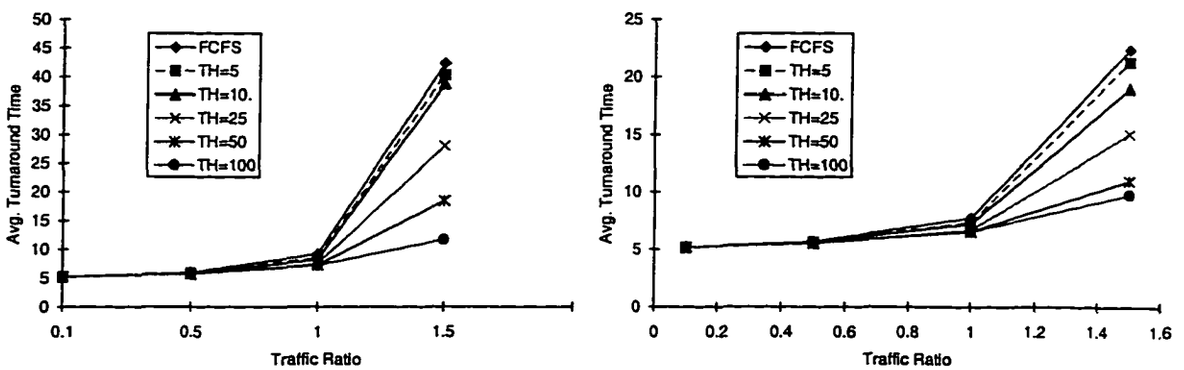
6.4 Performance Evaluation

Extensive simulations are conducted to evaluate the proposed FO allocation, the BQ scheduling, and the integrated strategies. The simulation parameters are the same as the ones used in Chapter 3. The BQ scheduling scheme is first evaluated with two

popular allocation algorithms, the first-fit and the adaptive-scan. The first-fit algorithm has no internal and very little virtual fragmentation. The adaptive-scan has only the insufficient resource and physical fragmentation. The BQ scheme significantly improves the performance of both of the allocation algorithms. The FO algorithm is then compared with the first-fit and the adaptive-scan to establish its feasibility. Results for the integrated processor management scheme which combines the FO allocation and the BQ scheduling schemes are also presented and discussed in this section.

6.4.1 Performance of the Bypass-Queue Scheduling

Figure 6.3 shows the effect of using the bypass-queue with different allocation schemes. Figure 6.3(a) is for the first-fit and Figure 6.3(b) is for the adaptive-scan algorithm. Different threshold time is considered for this comparison. The bypass-queue is equivalent to the ordinary FCFS queue when the threshold is set to 0. The bypass-queue scheduling efficiently reduces the average turnaround time for both allocation schemes. It also increases the operational range of the system. With the FCFS queue, the system gets saturated quickly for traffic above 1.5. With the bypass-queue, the curve is flattened and the system has a higher saturating point. Only the results for uniform job size distribution is shown. The normal job size distribution exhibits the similar behavior.



(a) First-Fit Algorithm (b) Adaptive-Scan Allocation

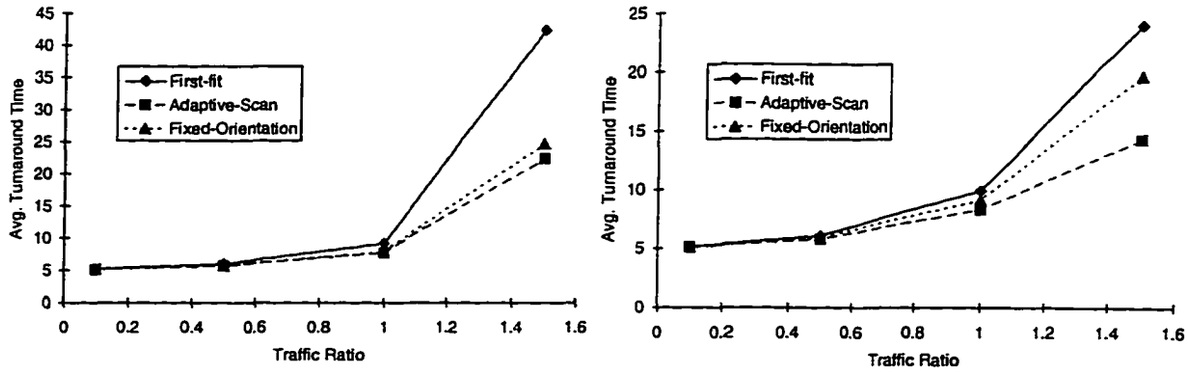
Figure 6.3 Effect of the Bypass-Queue Scheduling to Different Algorithms.

The bypass-queue scheduling works efficiently with the adaptive-scan allocation. A small increase in the threshold time results in significant reduction on the average turnaround time. It takes a larger threshold time for the first-fit algorithm to gain comparative performance improvement. The reason for this phenomena is the difference in the recognition abilities of submeshes between these two schemes. The BQ scheme allows jobs to bypass the blocking jobs for allocation and therefore reduces the average turnaround time of the system. All jobs that arrive between the submission of the first job in the queue and the expiration of the threshold time are candidates for the bypassing. Adaptive-scan has better submesh recognition ability than the first-fit algorithm and is more likely to successfully allocate a bypassed job. In order to reduce the average turnaround time with the BQ, the first-fit allocation needs a larger threshold time to allow more candidates to bypass.

6.4.2 Comparison among the Allocation Algorithms

The FO scheme is compared with the first-fit and adaptive-scan allocations. Figure 6.4 illustrate the comparisons with the uniform job size distribution and the normal job size distribution, respectively. The FO scheme outperforms the first-fit algorithm as expected because of the reduced virtual fragmentation by allocating jobs only in a fixed orientation. It provides shorter turnaround time than the first-fit algorithm for all traffic ratios. The average turnaround time is reduced by as much as 42% from the first-fit algorithm at traffic ratio of 1.5 in Figure 6.4(a).

The adaptive-scan performs slightly better than the FO scheme because of the absence of virtual fragmentation. However, the nearly identical performance of the FO allocation indicates that a good processor allocation scheme does not necessary need to eliminate all kinds of fragmentation. By properly arranging the allocation of jobs, fragmentation of the system can be avoided. The adaptive-scan gains its performance at the price of a higher computational complexity. Under low traffic, the FO and the



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

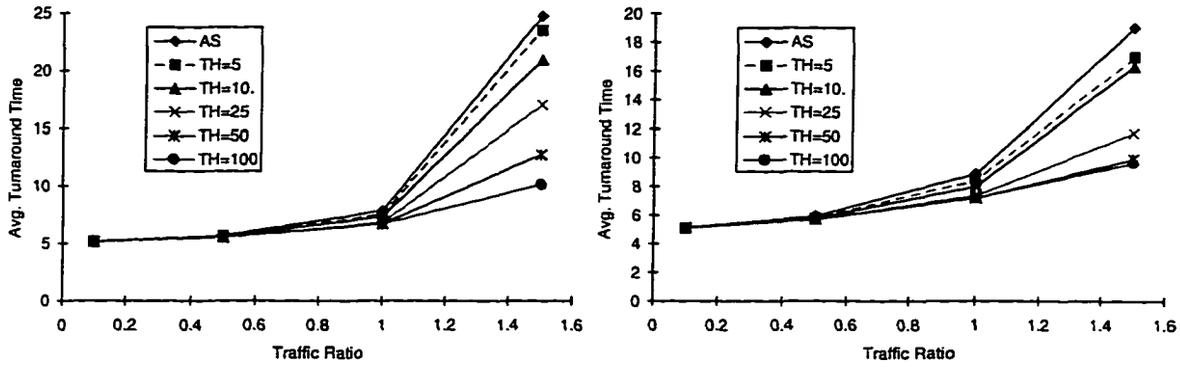
Figure 6.4 Average Turnaround Time of System using Different Allocation Algorithms.

adaptive-scan performs equally well with negligible difference. At higher traffic ratio, the adaptive-scan starts to perform slightly better than the fixed-orientation scheme. This is because in a heavily loaded system, jobs arrive and depart more frequently and the virtual fragmentation may be more serious. The adaptive-scan solves virtual fragmentation by checking alternative orientation for submesh allocation and therefore takes more time to perform. Because the allocation time spent depends on the environment where the allocation process is executed, it is hard to make a clear comparison. A rough estimate on the difference between the execution time of both algorithm is obtained. In a typical simulation run on the HP-9000/715 workstations with very little interference from other processes, the adaptive-scan takes about 25% more time to complete. The fixed-orientation allocation is feasible to be used in an integrated processor management policy because of its low computation demand.

6.4.3 Performance of the Integrated Policy

The integrated processor management scheme is simulated and the results are in Figure 6.5. The lines labeled *AS* is the average turnaround time obtained for the adaptive-scan algorithm using FCFS queue. It is included for comparison because adaptive-scan

has the best submesh recognition ability. The integrated scheme delivers better performance than the adaptive-scan with a small threshold time. Increasing the threshold time further reduces the average turnaround time of jobs. Both the BQ scheduling and the FO allocation contribute to this performance improvement. Because of the less fragmentation of the FO allocation, more jobs can bypass other jobs in the queue and get executed.

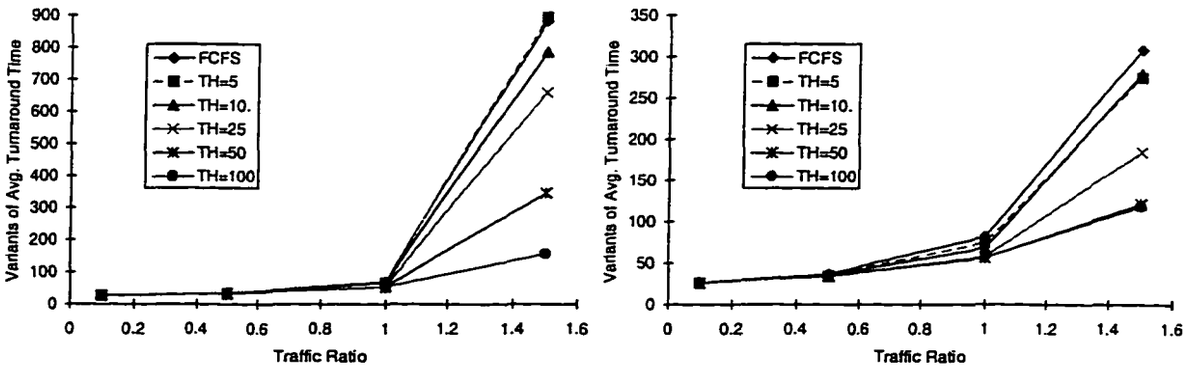


(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

Figure 6.5 The Integrated Processor Management Policy Using the Fixed-Orientation Allocation and Bypass-Queue Scheduling

Larger jobs have the tendency to be passed by other jobs with the BQ scheduling. Therefore, we compare the variance of the average turnaround time in Figure 6.6. Contradictory to our expectation, the variance does not increase when the threshold time increases. The variance is actually reduced when larger threshold is used. This is attributed to the fact that a larger threshold time reduces the turnaround time so efficiently that most of the jobs can be served within a short period of time and thus results in a small variance. A low variance for the turnaround time is a good property for the system because most of the jobs can be expected to finish within a certain range of the average.

Variance shows the square of the absolute distance from an expected value to the mean of the turnaround time. While a job with high turnaround time causes the variance

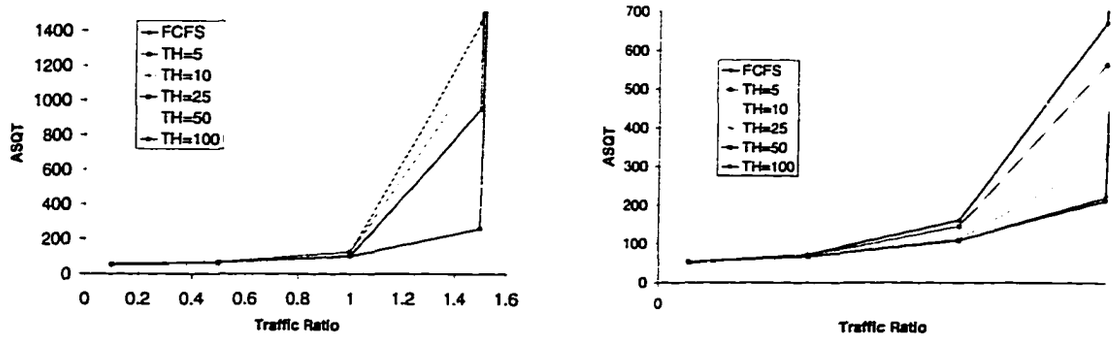


(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

Figure 6.6 Variance of the Avg. Turnaround Time for the Proposed Integrated Processor Management Policy.

to be large, a job with smaller turnaround time than the average also causes the variance to grow. In designing a processor management scheme, the main goal is to minimize the turnaround time of jobs. Penalizing a job with small turnaround time as the calculation of variance does hide the small turnaround time incurred by many jobs in the system. Therefore, we also use the average of the square of the turnaround time (shorthanded as *ASQT*) as a metric to compare the integrated policy with different threshold values.

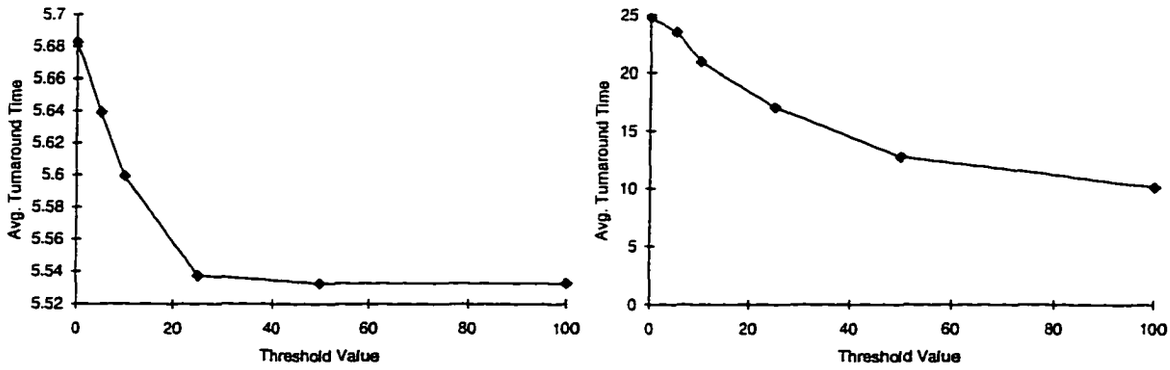
Using ASQT to evaluate the processor management schemes has two advantages. First, it amplifies the unfair treatment to some jobs. By taking the squares of the turnaround time of individual tasks, jobs suffered from unfair treatment from the processor management strategies will have more significant influence on the ASQT. Therefore, an unfair scheme is likely to have high ASQT value. The second advantage of using ASQT is to avoid penalizing schemes which results in jobs with relatively small turnaround time. Figure 6.7 illustrates the comparison among the integrated scheme with different threshold values. The trend shown is basically similar to that of the average turnaround time and the variance of the turnaround time. This observation confirms that the integrated scheme is not just efficient in reducing the average turnaround time of jobs in the system, it is also a fair policy.



(a) Uniform Job Size Distribution (b) Normal Job Size Distribution

Figure 6.7 ASQT for the Integrated Processor Management Policy.

Figure 6.8 demonstrates the effects of different threshold values on reducing the average turnaround time for two traffic ratios. A low traffic ratio of 0.5 is used in Figure 6.8(a) and a slightly high traffic ratio of 1.5 is used in Figure 6.8(b). Both cases are obtained from the uniform job size distribution, and the results for normal job size distribution exhibit the same trend. It is observed that the performance improves at a faster rate with the threshold time up to 25. The improvement is less for higher threshold values. This is especially true in the low traffic ratio case. As discussed earlier, the performance improvement of the bypass-queue scheme is obtained by allowing jobs to be allocated without being blocked by unallocated job. Number of jobs that can bypass a job is a function of the threshold time and the system load. Increasing the threshold time only improves the performance to a certain extent. On the other hand, using a large threshold time makes large jobs advance slow in the queue. From Figure 6.8, we conclude that a large threshold time is not necessary and suggest that a value in the range of twice the mean service time should provide fairly good performance.



(a) Low Load (Traffic Ratio = 0.5) (b) Medium Load (Traffic Ratio = 1.5)

Figure 6.8 Effect of Threshold Time on Reducing the Average Turnaround Time.

6.5 Discussion

The high complexities associated with existing processor allocation algorithms and job scheduling strategies makes the integration of the two approaches impractical. In this chapter, we have proposed a bypass-queue scheduling policy and a fixed-orientation allocation algorithm. The bypass-queue scheduling allows some jobs to bypass the blocked ones for execution and hence better utilizes the processors. The fixed-orientation algorithm allocates all jobs in a fixed orientation and thus avoids fragmenting the system. Both schemes have very low computational complexity and are therefore suitable for integration.

The bypass-queue scheduling reduces the average turnaround time for all the allocation algorithms we have tested. The fixed-orientation allocation performs better than the first-fit algorithm and is almost identical to the adaptive-scan algorithm. Trying all possible locations for allocation may not be necessary for good performance. Properly arranging jobs at allocation time to avoid future fragmentation can also improve the system performance as exploited in the case of the fixed-orientation allocation. The integrated processor management scheme that combines these two schemes result in further performance improvement. The average turnaround time of the integrated scheme is

better than the adaptive-scan with a small threshold time. Choosing a proper threshold time is important for the integrated scheme. Our results indicate that a large threshold value is not necessary and suggest that a threshold time in the range of twice the mean service time should provide fairly good performance.

7 JOB MIGRATION APPROACH

7.1 Introduction

In Chapter 2, we have discussed the fragmentation problem associated with the multicomputer systems. Fragmentation prevents the useful computing resources in the system from being utilized by incoming tasks. Various allocation schemes have been proposed to tackle different fragmentation problem. Most of the contemporary allocation schemes have the ability to eliminate the internal and virtual fragmentation completely. The ability to handle the insufficient resource fragmentation and the physical fragmentation becomes the defining point of an excellent allocation policy.

Insufficient resource fragmentation is caused by the insufficient computing resources in the system. Only the RSR scheme discussed in Chapter 3 has the ability to deal with this problem. It also has the ability to handle the physical fragmentation. The adaptive non-contiguous allocation method presented in Chapter 5 is the only other known method that has the ability to handle the physical fragmentation.

Both the RSR and ANCA schemes are non-conventional allocation schemes. Their feasibility for practical implementation depends on the characteristics of the job stream in the system. The effect of using users' input to implement processor allocation schemes will be discussed in Chapter 8. Another possible solutions for the fragmentation problems is by performing job migration. A job migration technique was proposed in [53] for the hypercube systems. Jobs are constantly migrated toward one end of the hypercube so that the available processors in the system are less fragmented. It has been proven to

be an effective way to manage the processors.

To our knowledge, no such schemes have been proposed for the mesh-connected multicomputers. In this chapter, we proposed a job migration approach for the mesh systems. The rest of this chapter discusses the proposed job migration algorithm and show the performance results of the proposed scheme.

7.2 Job Migration Process

The difficulty of performing job migration to improve the performance of multicomputer system in mesh lies in the irregularity of the sizes of jobs. In the hypercube-based system, the job sizes are distributed as subcubes. Any two subcubes of the same size can be put together and form a larger subcube and all nodes in the combined subcube will be fully utilized. Because of the regularity in subcube sizes, migrating jobs to one end of the system reduces the fragmentation and thus improve the system performance. In the mesh-based system, jobs come in the form of submeshes. It is more difficult to find two submeshes of the same size. Due to the high variances in submesh sizes, migrating a submesh to the side of another submesh does not necessarily reduces the fragmentation. Another problem associated with job size irregularity is the migratability of jobs. For hypercube systems, because jobs are all in the form of subcubes, it is easier to find an available subcube as the destination for the migrated job. For mesh system, it is more difficult to find a destination submesh for the migrating processes.

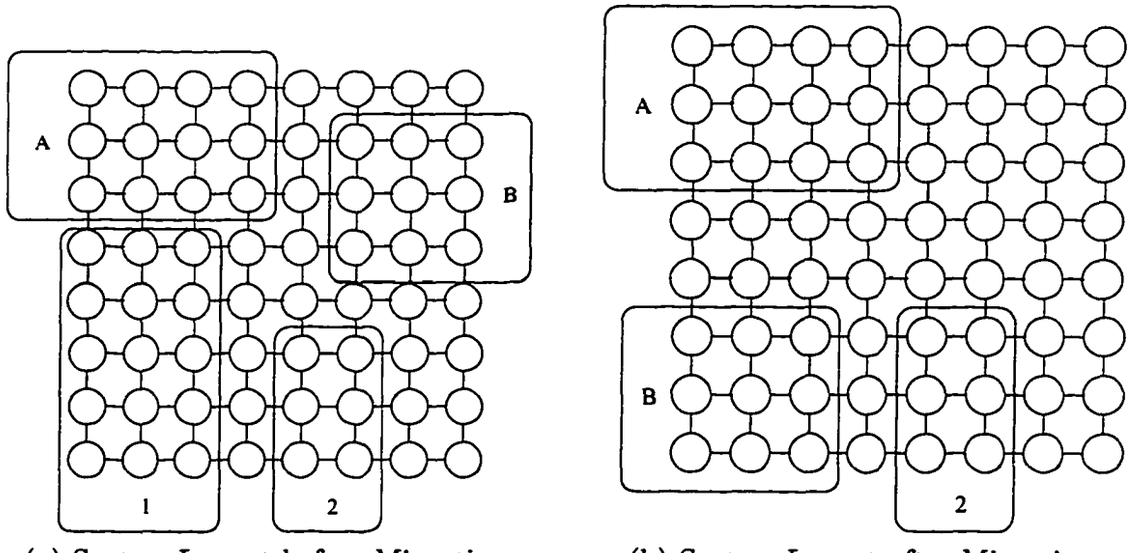
To overcome the problem of job size irregularity, we use the following heuristics to select a candidate submesh for job migration. First, a job which has an almost square shape is more likely to find a destination for migration. This is based on the observation made in the study of the fixed-orientation allocation scheme in Chapter 6. Jobs which have irregular sizes are more likely to cause the system fragmented and therefore reallocation of them should be avoided. Second, a small job is more likely

to be migratable. This is a straightforward observation from our previous study of allocation schemes.

The proposed job migration scheme works as follows. The information about most recently allocated jobs are maintained in a candidate pool. Job migration can be done at an failed allocation attempt or the departure of an existing job. When an allocation failed, a candidate for migration is selected from the candidate pool and the possible new location of this job is determined using the existing allocation algorithms. By carefully arranging the allocation algorithm, the new location of the job can be made to be closer to one corner of the system. The job that failed to be allocated will be reevaluated for the allocation in the system under the assumption that the candidate has been moved to the new location. The candidate is migrated only when the migration enables the allocation of the previously unallocated job. This approach is referred to as the *arrival* approach. Job migration can also be done at the departure of an existing job. When a job terminates and leaves the system, it release all the processors it holds. The newly released nodes could be surrounded by other executing jobs and therefore creates a fragmented hole of processors in the system. An aggressive approach would be to migrate jobs when the number of free processors in the system changes. In the case of migration on job departure, the same candidate pool is used for the selection of migration candidates. Upon completion of a job, the system selects a job from the candidate pool and check if it can be migrated. If a new location can be found, the candidate job is migrated. Again, the new location of the job is so decided that it is closer to one of the four corners of the mesh. It can therefore be imagined as having the allocated jobs drifting to one side of the system upon the completion of jobs. This approach is referred to as the *departure* approach. An even more aggressive approach is to perform job migration at both the completion of a job and a failed allocation attempt which is referred to as the *aggressive* approach in the rest of this chapter. To minimize the cost of job migration, we only migrate a job every time the migration is

initiated. The migration pool is used to simplify the candidate selection process. To avoid migrating a job multiple times and causes its execution to be penalized multiple times, any job can only be migrated once.

Figure 7.1 illustrates how the jobs are migrated. In this example, four jobs are executed in the mesh as shown in Figure 7.1.(a). Assuming a candidate pool size of two jobs and the two jobs in the candidate pool are labeled as job A and B. When job 1 leaves the system, job B is selected as the migration candidate because of its size and shapes meets our heuristics for candidate selection. The allocation algorithm then tries to find a new location for job B as close to the lower-left corner as possible. As a result, the new allocation of the three remaining jobs ends up as illustrated in Figure 7.1.(b). If an on-demand migration approaches is used, job B will still be selected as the candidate. The allocator checks if the allocation of the incoming task is possible assuming job B has been moved toward the lower-left corner. If the new job can be allocated by moving job B, job B is migrated and the new job is allocated. Otherwise, job B remains intact and the new job has to wait until the departure of an executing job for allocation.



(a) System Layout before Migration (b) System Layout after Migration

Figure 7.1 Example of an Aggressive Job Migration Approach.

7.3 Performance of the Job Migration Schemes

The performance of the job migration scheme is evaluated in this section. Three migration approaches, *arrival, departure* and *aggressive* migration are evaluated. The cost of migration is very difficult to estimate. It depends on the size of the job to be migrated, the speed of interconnection network, and the ability of the system to handle the suspension and restart of the migrated process.

The speed of the interconnection network used in most of the contemporary machines are typically around tens or hundreds of megabits per second. This means the actual migration process of a job may take only a few seconds or even less even for a large job. Compared to the run time of typical parallel application on multicomputer systems, this cost is insignificant. However, migrating a job does require the support from the system. For example, the operating system has to identify the new location and move the contents of the register file and the memory pages to the new processor. It is also important to make sure all the messages generated by a job which is about to be migrated reach their destination before the migration process takes place. Otherwise, a stranded message may never reach its destination and cause the job to wait for it indefinitely. It is also possible for a message from a migrated job to reach a incorrect destination processor after that processor has been assign to another job and causes the execution of the new process to fail. The solution to these problems is by inserting checkpoints in the programs. Checkpoints can be inserted manually by the programmer or automatically by the system. When a job is migrated, we can either resume its execution on its new location from the most recent checkpoints or wait till the next checkpoint before migrating it. Waiting for the next checkpoint may not be useful in the system we are considering because the lack of information on job execution time also implies lack of information on the time to next checkpoint. By going back to the previous checkpoint, some amount of computation is lost and has to be taken into account when designing a

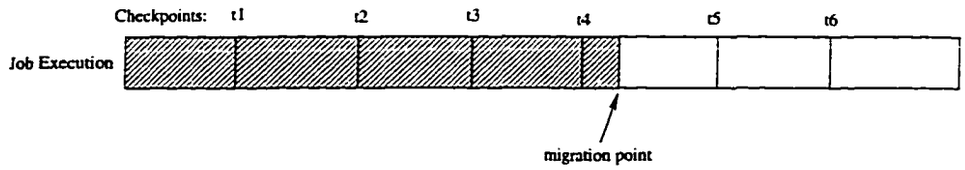


Figure 7.2 Checkpoints and the Job Execution.

job migration scheme.

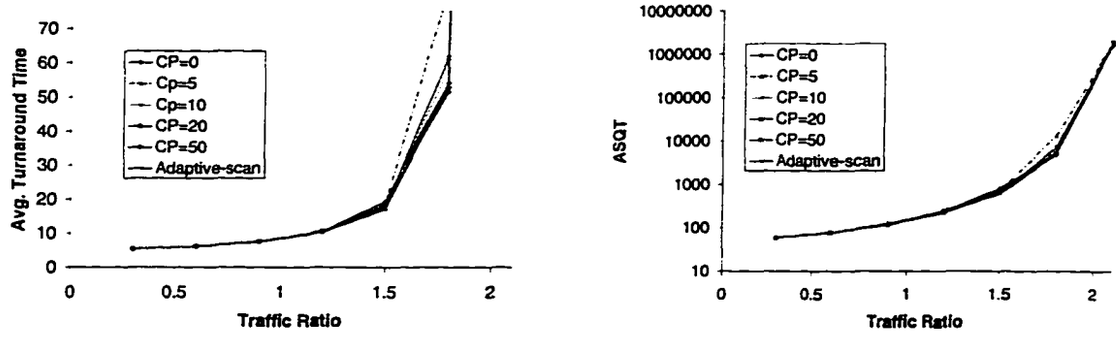
In our performance study, we assume the system is capable of restarting a job at the most recent checkpoint. Figure 7.2 shows the execution of a job which has six checkpoints. The shaded area represents the completed execution. At the time of migration, execution has passed checkpoints t4 and has yet to reach point t5. The execution has to resume from point t4 and thus results in a migration penalty of losing the execution between point t4 and the migration point. We studied several different cost factors of job migration based on the number of checkpoints in a job. The cost of transferring the code and data from original location to the newly allocated nodes is ignored. Once a job is migrated, its termination time is delayed by its relative position to the previous checkpoint.

7.3.1 Performance of Individual Migration Approaches

The first set of results illustrates the difference between different migration approaches. The two performance metrics shown are the average turnaround time and the average of the square of the turnaround time. The average turnaround time is a direct measure to show the system performance from the user's perspective. It indicates the time that a user has to wait for his or her jobs to complete after the submission. Lower average turnaround time also indicates a higher system throughput. The average of the square of the turnaround time, shorthanded as $ASQT$ is an indication of the fairness of a scheme. Due to the differences of the processor management schemes, some jobs may be treated unfairly in order to achieve the overall system performance. When a job

is treated unfairly, its turnaround time is relatively larger than most other jobs. Taking the average of the square of turnaround time of all jobs amplifies the high turnaround time incurred by jobs treated unfairly. A good processor management scheme should result in low average turnaround time while a fair system should have low ASQT.

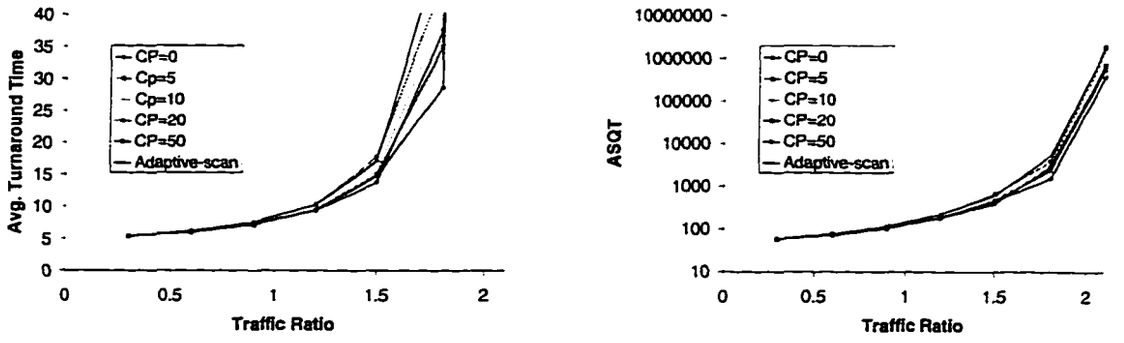
Figures 7.3, 7.4 and 7.5 are the results obtained when migration is applied at job allocation, job departure and both job allocation and departure for the uniform job size distribution case. Each line in the figures represents a different number of checkpoints. The line labeled $CP = 0$ is the case when the system can migrate a job at any instant without losing any execution and therefore does not require any checkpoints. Similar results are shown in Figures 7.6, 7.7 and 7.8 for the normal job size distribution. The size of the candidate pool used in these simulations are six.



(a) Avg. Turnaround Time (b) ASQT

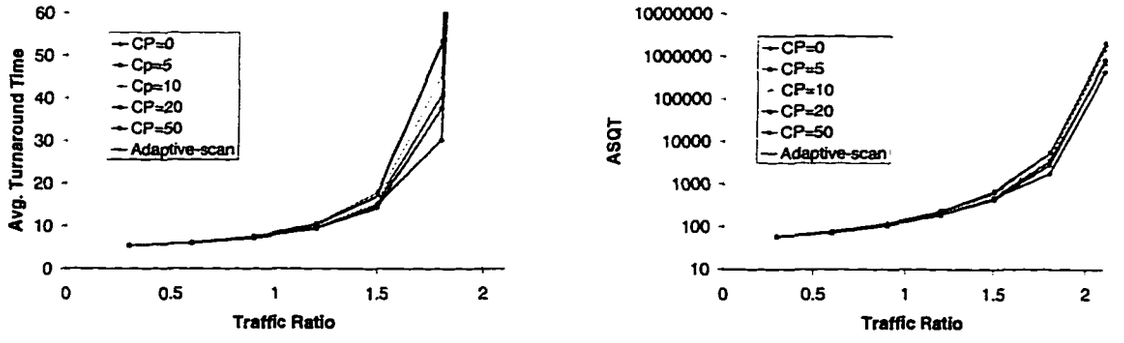
Figure 7.3 Migration at Job Allocation (Uniform Jobs).

It is interesting to see the on-demand approach of performing job migration upon an allocation failure does not provide better performance than the adaptive-scan allocation. In many cases, it provides longer average turnaround time than the adaptive-scan allocation. This can be explained as the inefficiency of the on-demand approach. Because we only migrate one job at a time, the on-demand approach does not guarantee a less fragmented system. With the cost of restarting execution from previous checkpoints, the



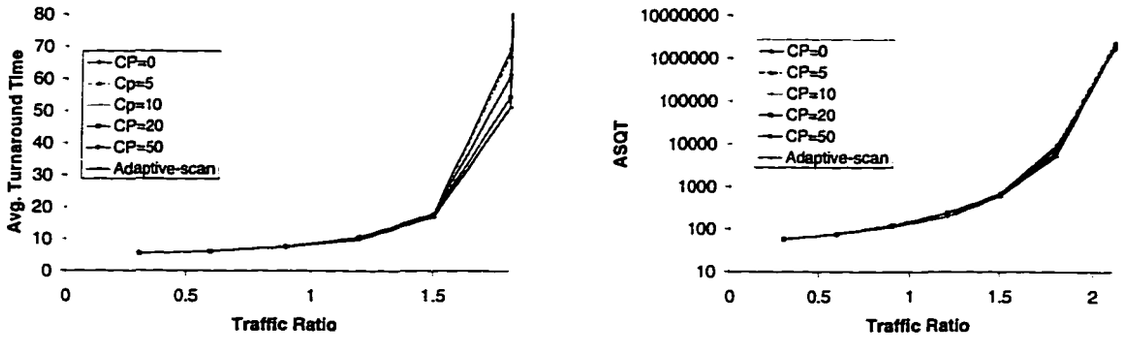
(a) Avg. Turnaround Time (b) ASQT

Figure 7.4 Migration at Job Departure (Uniform Jobs)



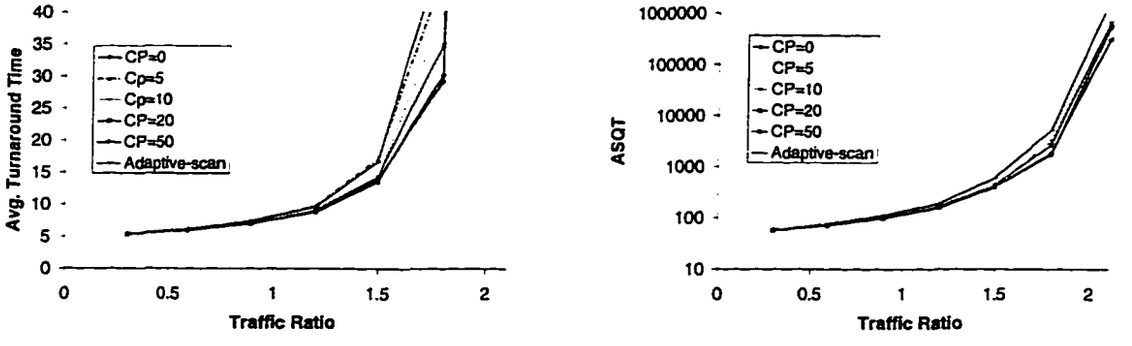
(a) Avg. Turnaround Time (b) ASQT

Figure 7.5 Aggressive Migration at Both Job Allocation and Departure (Uniform Jobs).



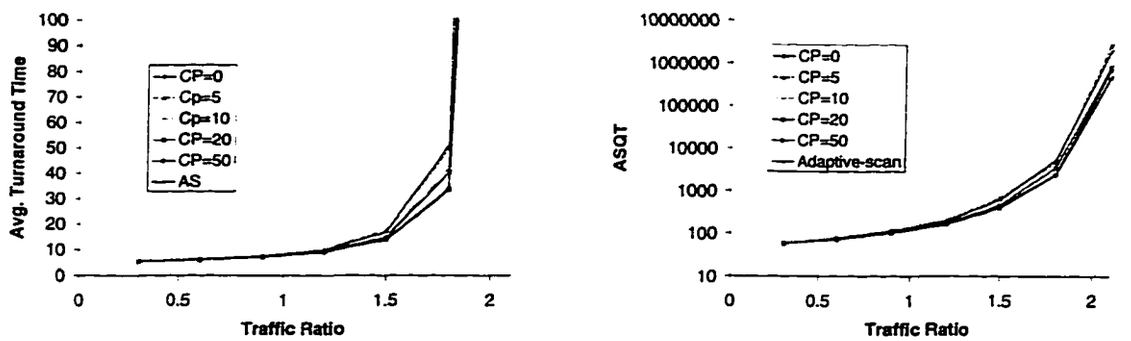
(a) Avg. Turnaround Time (b) ASQT

Figure 7.6 Migration at Job Allocation (Normal Jobs).



(a) Avg. Turnaround Time (b) ASQT

Figure 7.7 Migration at Job Departure (Normal Jobs).



(a) Avg. Turnaround Time (b) ASQT

Figure 7.8 Aggressive Migration at Both Job Allocation and Departure (Normal Jobs).

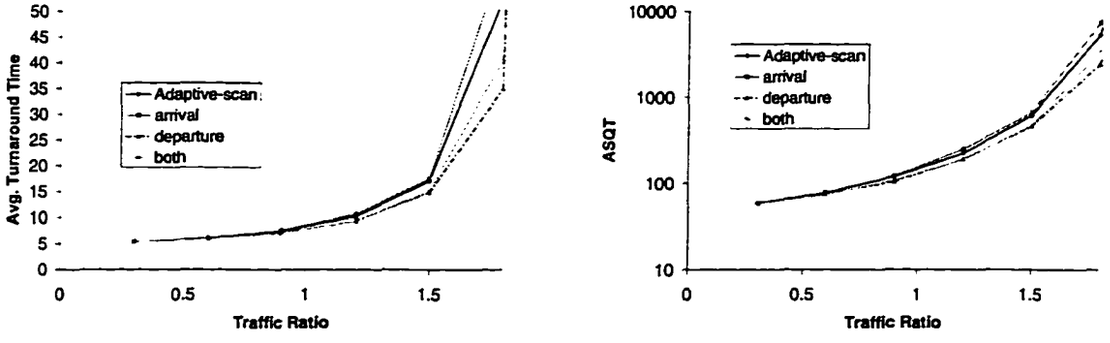
on-demand approach fails to deliver a better performance. Both the *departure* and *aggressive* approaches outperforms the adaptive-scan provided the number of checkpoints in the job is sufficient. This is due to the fact that by migrating jobs aggressively toward one corner of the system, fragmentation is reduced and hence the queuing delay is reduced. Having little checkpoints in a program means a high restarting cost for migrated jobs and therefore cause the job migration approach to produce high average turnaround time. It is observed that if the number of checkpoints in a job is more than five, the average turnaround time of jobs benefits from performing *departure* or *aggressive* migration.

The average of the square of the turnaround time is a metric which indicates the fairness of the processor management schemes. If a job is treated unfairly, it will incur high turnaround time and therefore cause the average of the square of the turnaround time for the system to be high. In our results, when the number of checkpoints is equal to two or five, this value is high indicating that some jobs are treated unfairly because of the migration cost. With more checkpoints in a job, the value is lower than the adaptive-scan indicating that jobs are treated fairly and most jobs experience smaller

turnaround time.

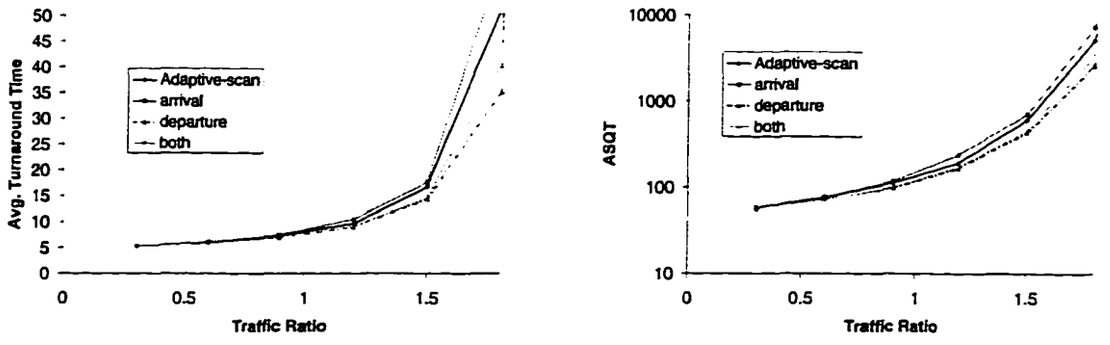
7.3.2 Comparison Among Migration Approaches

The three job migration approaches are compared in Figures 7.9 and 7.10. The results indicate that the *departure* approach has the best performance among the three. The *arrival* approach is the worst scheme. Both the uniform and normal job size distribution confirm this observation. The arrival approach migrates jobs upon allocation failure and perform the migration only when the unallocated job can be allocated after migration. The available processors in the system when allocation failure are limited and therefore limit the flexibility of job migration. To reduce the fragmentation effectively, jobs have to be migrated to one end of the system as much as possible. Upon job departure, the number of available processors are more and giving more flexibility to migrate a job. The aggressive approach performs between the departure and arrival approach. This is because some of the jobs in the candidate pool are migrated at the allocation failure of other jobs and thus negate the effect of migrating at departure.



(a) Avg. Turnaround Time (b) ASQT

Figure 7.9 Comparison among Migration Approaches (Uniform Jobs).



(a) Avg. Turnaround Time (b) ASQT

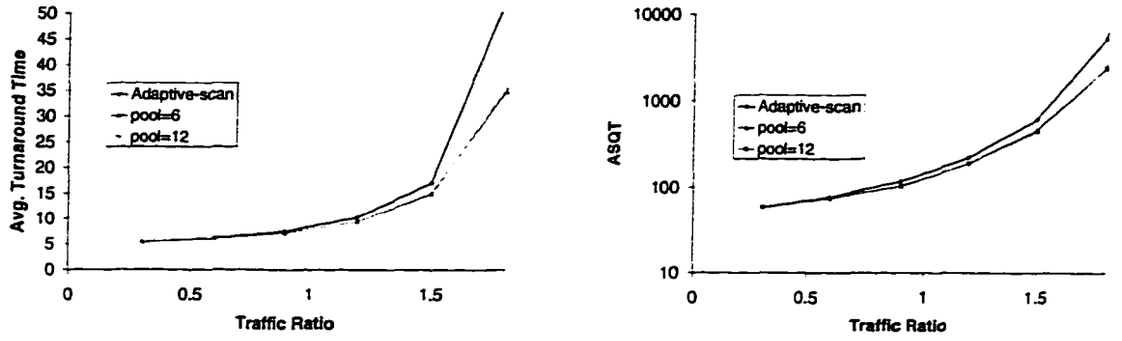
Figure 7.10 Comparison among Migration Approaches (Uniform Jobs).

7.3.3 Effect of Candidate Pool Size

All the results shown in previous sections are obtained with a candidate pool of six jobs. A larger pool size is likely to provide a better candidate for migration and may perform better. Different candidate pool size are compared in Figures 7.11 and 7.12. Based on the observation from the previous section, only the departure approach is compared. The difference between a pool size of six and a pool size of twelve are very limited, contractdicting our expectation. Both lines overlap with one another. This is true for both job size distributions and the two metrics compared. Using a larger candidate pool also increase complexity of the candidate selection process. Even though the complexity of candidate selection process is low, the nearly identical performance does not suggest the use of a larger candidate pool.

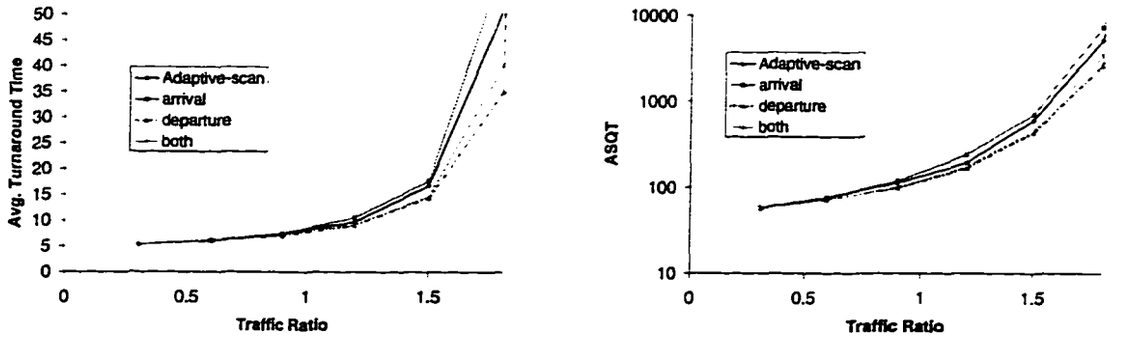
7.4 Discussion

This chapter discusses an alternative processor management scheme, the job migration approach. Three different migration approaches are evaluated and the effect of the size of the candidate pool is also studied. Our simulations indicate that migration



(a) Avg. Turnaround Time (b) ASQT

Figure 7.11 Effect of Candidate Pool Size (Uniform Jobs).



(a) Avg. Turnaround Time (b) ASQT

Figure 7.12 Effect of Candidate Pool Size (Uniform Jobs).

upon completion of a job provides the best performance. The difference among the job migration approaches is the flexibility for selecting a new location for the migrating jobs at the time migration is performed. Migrating upon the completion of a job allows the migration process to take advantage of the free processors released by the departure job. Increasing the size of the candidate pool does not further improve the system performance.

Another factor that has to be taken into consideration when implementing the job migration approach is the cost of migration. In our simulation, the cost for transferring the data and program is ignored. This may not be the case in a real implementation and has to be carefully considered. Also the cost of restarting the job from a previous checkpoint cannot be ignored as seen in our results. The support of checkpointing from hardware and/or operating system has to be implemented in order to take advantage of the job migration approach.

8 USING USER DIRECTIVES FOR PROCESSOR ALLOCATION

8.1 Introduction

In previous chapters, we have discussed various processor management techniques in the multicomputer systems. Two techniques, the RSR and the ANCA schemes have shown promising performance improvement comparing to the other techniques. However, the potential problem associated with the penalty of using them limits the practical use of these two novel approaches. The penalty involved when using the RSR scheme is the extended execution time of jobs being allocated on size-reduced submeshes. The penalty involved in the ANCA scheme is the increased communication latency of jobs being allocated non-contiguously. Both these penalty can be reduced if user directives can be incorporated into the design of the processor management policies.

In an environment where not all jobs can be allocated to less processors or to non-contiguous nodes, the penalty for performing RSR or ANCA allocation may be too high and causes poor overall system performance. In such an environment, conventional allocations which conservatively allocate all jobs to contiguous processors according to the job request may be used to avoid the unnecessary penalty associated with the RSR or ANCA schemes. If a job is known to require very few interprocessor communication, the penalty incurred by allocating this job to non-contiguous processor is less likely to affect the overall execution time of it. The performance of the whole system may actually benefit without paying much of a penalty for doing non-contiguous allocations of this

kind of jobs. Similarly, if the memory requirement of a job is not tight, then it is unlikely to exhaust the available memory in each processing unit when allocated using RSR. The system-wide performance can again benefit from this information. On the other hand, if a communication-intensive job gets allocated non-contiguously or a memory-bounded job's size gets reduced, the penalty may be higher than what we expected. Therefore, if information about certain characteristics of a job can be used to assist the allocation, a higher performance of the system can be expected. In this chapter, we discuss the effect of having user specify the characteristics of jobs for the allocation process.

8.2 User Directives for Processor Allocation

Conventional allocation schemes have reached a performance bottleneck. The performance difference among various allocation algorithms are minimal because of the size and shape constraints imposed by the conventional allocation algorithms. Many researchers have therefore resorted to job scheduling for efficient processor management. This chapter discusses a new processor allocation schemes which combines three very different allocation approaches with the assistance of user directives to take advantages of each allocation approaches.

The concept of size-reduction has been used in the RSR allocation. Non-contiguous allocation has also been implemented in the ANCA scheme. Both size-reduction and non-contiguous allocation are radical changes from the conventional processor allocation approaches and have been proven to provide significant performance improvement. An ideal processor management policy would have included these two novel approaches. However, these two approaches are subject to some performance limitations.

When size-reduction is applied to a job, the amount of computation executed on each processor is increased due to the smaller number of processors allocated to the job. This increases the execution time of the job. Another issue needs to be addressed for

size-reduction schemes is the memory requirement. The data required in the execution of a job is redistributed among a smaller number of processors if size-reduction has been applied to the job. A potential problem can happen if the memory required for running the job exceeds the memory threshold of the processors. This could cause the system to fail or cause the job to suffer from memory swapping and further increases the job execution time. Therefore, memory requirement is an important factor to consider when implementing the size-reduction schemes.

The performance limitations associated with the non-contiguous allocation algorithms is the potential increase of the communication latency experienced by jobs being allocated non-contiguously. Parallel applications and algorithms are optimized to minimize the number of communication steps and the distance of communication path. Non-contiguous allocation violates the optimized communication path and may cause the communication latency of jobs to increase. The execution time of the job is therefore increased with the increasing communication latency. Studies in [55] indicated that substantial increase of the job execution time can be observed if non-contiguous allocation is used. The increase of job execution time is especially pronounced for communication intensive jobs.

The penalty of implementing size-reduction or non-contiguous allocation can be avoided if certain characteristics of the job requests is known a prior. Two attributes of jobs' characteristics, communication intensity and memory requirement, are particularly useful in implementing processor management schemes. If a job is known to be communication intensive, non-contiguous allocation has to be avoided. On the other hand, if the communication demand of a job is known to be low, the system can take advantage of this fact and allocates it non-contiguously to increase the processor utilization. If a job is known to be memory-bounded, size-reduction should not be taken. If a job requires little memory or if its turnaround time is not a critical consideration, size-reduction may increase the possibility of its allocation and improve the overall system performance.

In this chapter we analyze the use of user directives as an assistance to processor allocation. Instead of using a single allocation scheme for all jobs, a combination of three different allocation schemes are used based on the attributes of the job being allocated. The three allocation schemes considered are the conventional algorithm, the RSR scheme, and the ANCA policy. The proposed allocation scheme is referred to as *hybrid allocation*. The conventional algorithm allocates jobs to a set of contiguous processors based on the requests. The RSR scheme allocates jobs to a size-reduced submesh when the submesh originally requested cannot be located in the system. The ANCA policy adaptively allocates jobs non-contiguously when fragmentation prevents jobs from being allocated. Both RSR and ANCA schemes allocates jobs to requested submesh whenever possible so that the penalty from size-reduction or non-contiguous allocation is minimized. Table 8.1 lists the combinations of the job attributes and the possible allocations that these combinations implied. With the assistance of the user directives, it is expected that the system can take advantages of each allocation scheme. For jobs which are communication intensive and memory bounded, the conventional allocation algorithm is used. Jobs which has more relaxed memory requirement can be considered for the RSR allocation. Non-communication-intensive jobs can be considered for the ANCA scheme. Jobs that are neither communication-intensive nor memory-bounded can be considered for all allocations.

Table 8.1 Job Attributes and Possible Actions

	Communication-Intensive	Non-Communication-Intensive
Memory-Bounded	Conventional only	ANCA
Non-Memory-Bounded	Conventional or RSR	ANCA or RSR

The *hybrid allocation scheme* using user directives is described as following. Upon submission of a job into the system, users can specify the two attributes for the jobs, whether it is memory-bounded or communication-intensive by setting the associated flags. Upon the allocation of a task, the flags are checked and the allocation scheme

is chosen based on the attributes of the job. If a job can be allocated by both RSR and ANCA scheme, RSR is used because of its ability to fit more jobs into the system. Setting of the flags requires the knowledge about the job characteristics. It is assumed that the users of multicomputer systems are experienced scientists and therefore should be able to determine the two flags for their submissions. However, it is possible for the users to not know the characteristics of their jobs or to provide false information. By default, a job is assumed to be memory-bounded and communication-intensive unless otherwise noticed by the user. This assumption is to avoid false treatment of a job and hence bad performance.

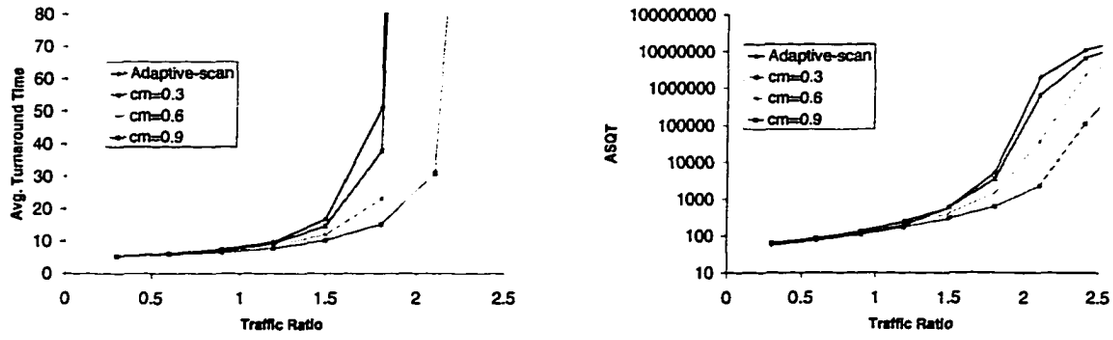
8.3 Performance of Hybrid Allocation Using User Directives

The processor allocation scheme using user directives is evaluated with simulation. The default simulation parameters such as system size and job size distribution are the same as the one described in Tables 3.1 and 3.2. The results are the average of 10,000 jobs over multiple repetitions. In each iteration, the first 1,000 jobs are excluded from the data collection to avoid premature data.

We simulate the allocation with user directives using two independent variables for the characteristics of a job. The variable *cm* is the ratio of jobs that is communication-intensive. The variable *mm* is the ratio of jobs that is memory-intensive. By varying the these two variables, we expect to see the system's ability to handle different mix of jobs. With the assistance of user directives, it is safe to assume that the jobs get allocated using ANCA scheme or RSR scheme are not communication-intensive or memory-intensive. Therefore, for the ANCA scheme a cost factor of 5% is assumed for jobs allocated non-contiguously. The execution time of jobs allocated non-contiguously are assumed to be increased by 5% to reflect the possible increase in communication latency. For the jobs get allocated non-contiguously, their execution time is assumed to be doubled for

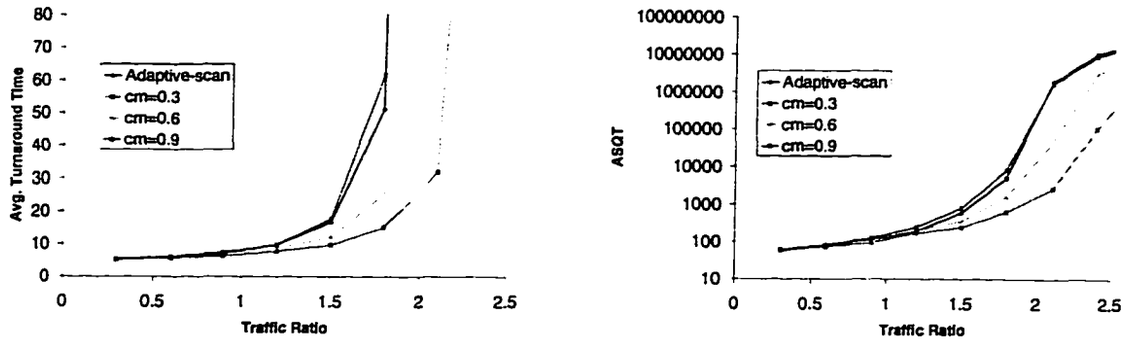
every size-reductions they incurred as assumed in Chapter 3. Knowing a job is not memory-intensive ensures its feasibility for size-reduction and memory thrashing is not likely to occur. However, the amount of computation in such a job remains about the same so its execution time is still assumed to increase exponentially with the number of size-reductions it incurred. As indicated in Chapter 3, a size-reduction of two to four is effective enough to improve the system performance dramatically, we use the RSR-2 in our evaluation. As of ANCA, because of the low cost associated with it when user directives are used, we use ANCA-10 allocation.

In Figures 8.1 to 8.3, we show the two performance metrics for memory-intensive ratio at 0.3, 0.6 and 0.9 for the normally distributed jobs. Higher values of cm result in higher values in both metrics measured. This is because the lack of flexibility in choosing jobs to perform ANCA or RSR allocation. It is observed that in most cases, using user directives does provide better performance than using the conventional adaptive-scan allocation. The possibility to take advantages of the ANCA and RSR scheme proves to be effective. However, when $mm = 0.6$ and $cm = 0.9$, using user directives actually resulted in worse average turnaround time and average of the square of the turnaround time.



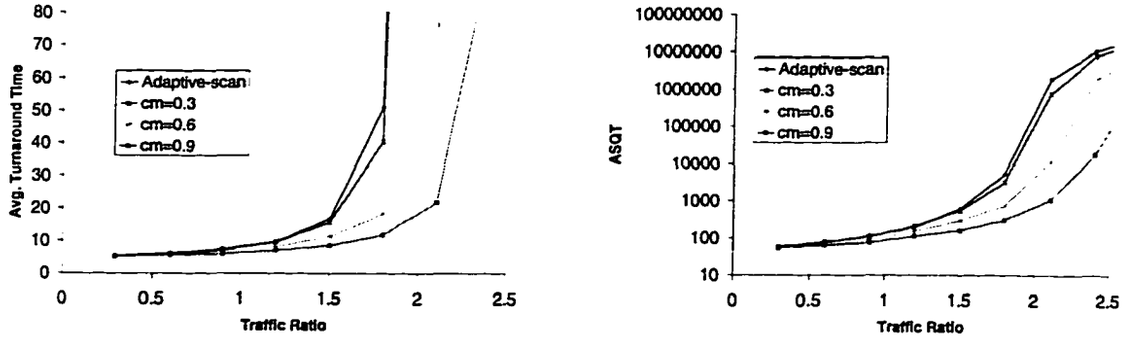
(a) Avg. Turnaround Time (b) ASQT

Figure 8.1 Using User Directives (Normal Jobs, $mm = 0.3$).



(a) Avg. Turnaround Time (b) ASQT

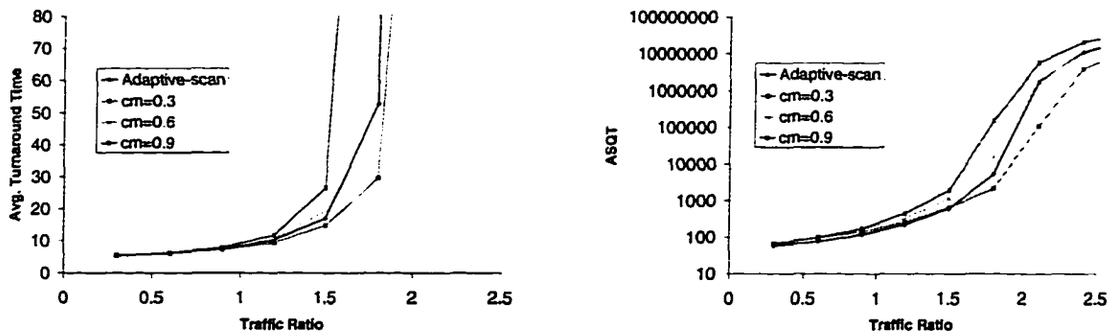
Figure 8.2 Using User Directives (Normal Jobs, $mm = 0.6$).



(a) Avg. Turnaround Time (b) ASQT

Figure 8.3 Using User Directives (Normal Jobs, $mm = 0.9$).

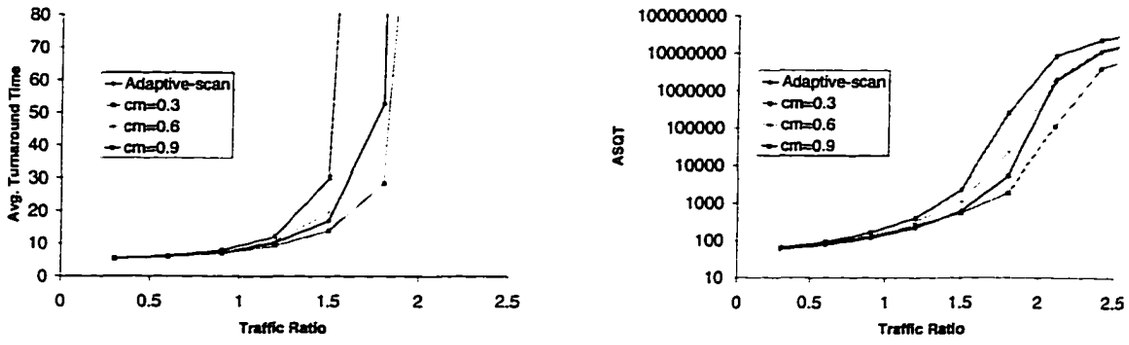
The results for the uniformly distributed jobs are shown In Figures 8.4 to 8.6. Using user directives fails to improve the system performances in many cases. In most cases, only when the ratio of communication-intensive jobs is 0.3, the allocation using user directives can take advantages of the two novel schemes. When the ratio of memory-intensive jobs equals to 60% (Figure 8.5), the proposed approach are particularly bad compared to the adaptive-scan allocation. The observation made from these simulation is interesting in the sense that it contradicts the expectation that when user directives are incorporated into the allocation, the system can take advantage of the RSR and ANCA schemes and reduce the average turnaround time of jobs. Another problem associated with the processor allocation using the user directives is the unfairness treatment to the jobs. As measured in the simulations, using user directives often results in high ASQT values indicating that some of the jobs indicating considerably higher turnaround time than the others. This is a character of the system which is not desired.



(a) Avg. Turnaround Time (b) ASQT

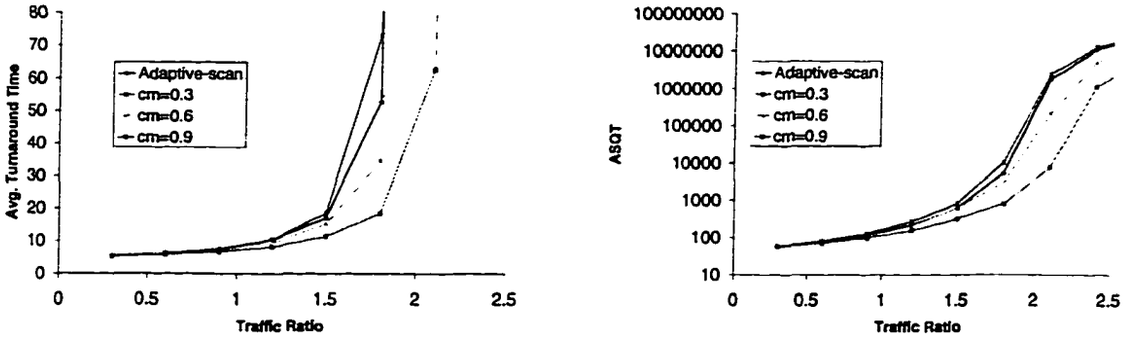
Figure 8.4 Using User Directives (Uniform Jobs, $mm = 0.3$).

The reason for this performance drawback of the hybrid allocation is a combination of the high cost associated with the RSR scheme and the lack of flexibility in choosing jobs for performing size-reductions. In the simulation of the RSR scheme in Chapter 3, the allocator has complete freedom in performing size reduction when fragmentation



(a) Avg. Turnaround Time (b) ASQT

Figure 8.5 Using User Directives (Uniform Jobs, $mm = 0.6$).



(a) Avg. Turnaround Time (b) ASQT

Figure 8.6 Using User Directives (Uniform Jobs, $mm = 0.9$).

prevents a job from being allocated. The blocking effect of the queue is therefore alleviated when the jobs can be allocated to smaller size submesh. Without the freedom of performing size-reduction on every job, the high cost associated with size-reduction becomes a severe factor in the queuing latency. For example, in Figure 8.7, a job is allocated in the center of the mesh with size-reduction. Due to the size-reduction, its execution time is considerably higher than its original execution time. If the next job in the queue which requires a 4x4 submesh for its execution is labeled as memory and communication-intensive, RSR or ANCA cannot be used and it has to be allocated using conventional allocation scheme. The 4x4 job will then wait for the departure of the jobs in center of the system before it can starts execution. Consequently all jobs after it have to be queued and cause the poor performance of the system.

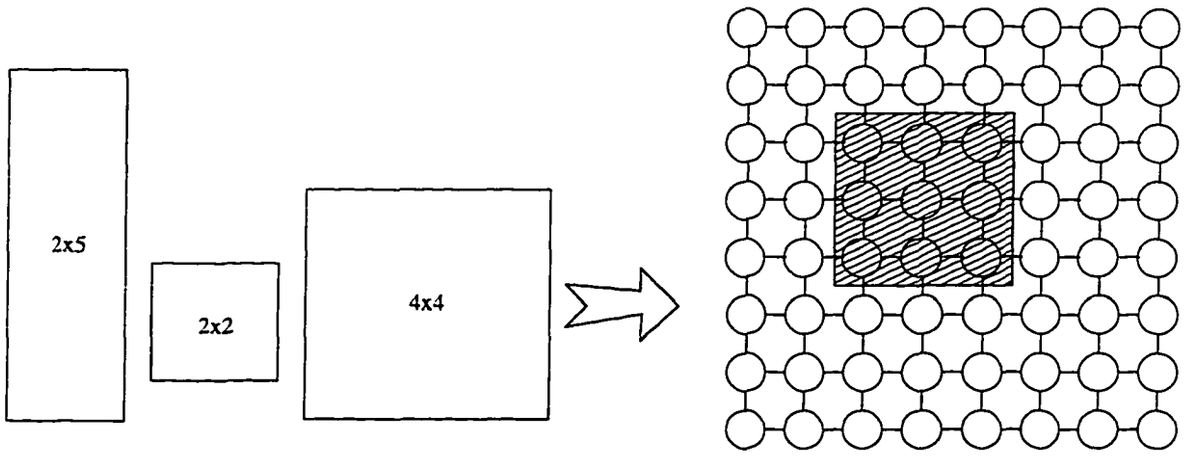


Figure 8.7 Blocking Caused by a Job Allocated Non-Contiguously.

A combination of a size-reduced job following by a blocked job which cannot be allocated with RSR or ANCA cause the system to produce high average turnaround time. Large jobs are more likely to be blocked or to go through size-reduction. In the normal distribution case, job sizes are more concentrated to about half of the system size in each dimension. For the uniform distribution, job sizes are more diversified over the possible range and hence have more large jobs in the system. It is the reason that

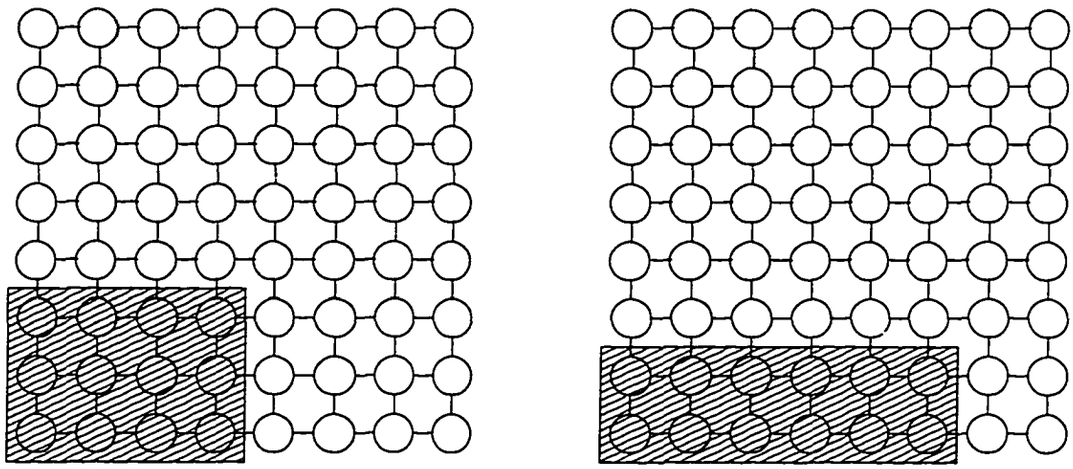
the uniformly distributed case has worse performance than adaptive-scan when the user directives are used. The jobs allocated with size-reduction have to be carefully placed in the system so that they do not create fragmentation and cause blocking of other jobs.

8.4 Modified RSR

The RSR allocation is shown to have performance concerns in an environment where not all jobs can go through size-reduction. As the performance study in the last section indicates, due to the lack of freedom on performing size-reduction on every job possible and the location of jobs allocated non-contiguously, serious blocking may occur. The original RSR algorithm does not have this problem because all jobs are assumed to be non-memory-intensive and can therefore avoid the blocking by performing size-reduction. To solve the problem of RSR allocation in an environment where user directives are used, we propose a modified RSR algorithm.

The proposed modification of the RSR scheme takes advantage of size-reduction while avoids the blocking problem. The blocking problem for the scheme used in Section 8.2 is mainly caused by the long execution time of the size-reduced jobs. If such jobs are placed in the center of the mesh, many jobs will be blocked. This problem can be solved if the jobs allocated with size-reduction are carefully placed along the periphery of the mesh so that they do not block other jobs. To further reduce the possibility for a size-reduced job to cause fragmentation in the system, the decision on the dimension to fold is also modified. The original RSR scheme folds a job in the larger dimension when folding is necessary. The rationale behind this decision is to make the allocated submesh more regular (closer to square) so that the job can be easier allocated and once it departs the system, the space it lefts in the system is more likely to be allocated to other jobs. Figure 8.8 illustrates two folding decisions and their impact on the fragmentation of the system. Suppose a job requiring 6x4 submesh is folded twice and placed at the bottom-

left corner of the mesh. Using the original RSR, a 3x4 submesh is allocated to this job as in Figure 8.8.(a). Another possible way of folding is to always fold a job along the same dimension as shown in Figure 8.8.(b). In this case, a 6x2 submesh will be allocated and the system is able to accommodate larger jobs compared to the case when 3x4 submesh is allocated. It is also important to notice that the orientation of the allocated submesh has to be aligned with the periphery so that fragmentation of the system is avoided.



(a) Folding in alternative sides (b) Folding along the same side

Figure 8.8 Comparison of the Folding Decisions.

The modified RSR is therefore described as follows. Upon allocation of a job, its original submesh requirement will be examined for allocation. If the allocation failed, the larger dimension of it will be selected as the dimension to fold. The folded submesh is checked along the periphery of the mesh for the possible allocation. Because the orientation of the submesh has to align with the periphery of the mesh, only one orientation of the submesh has to be checked along each edge of the mesh. If the allocation again failed, the submesh is folded in the same dimension and the checking continues until the preset limit of allowed size-reduction is reached.

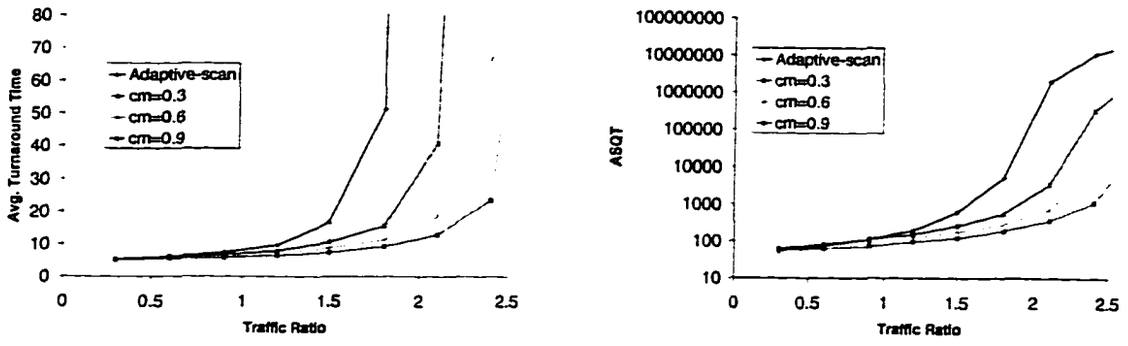
8.5 Performance of Processor Allocation Using Modified RSR with User Directives

The simulation results for processor allocation using modified RSR with user directives are shown in Figures 8.9 to 8.14. Significant reduction on average turnaround time is observed in all cases but one. The only exception is with uniform distribution, $mm = 0.6$ and $cm = 0.9$. In this case, the proposed processor allocation using modified RSR performs almost identical to the adaptive-scan allocation. Its average turnaround time is less than 0.5% more than the adaptive-scan. Comparing to the performance improvement in other cases, this difference is very insignificant. In addition, the percentage of jobs which are memory-intensive is likely to be higher than the percentage of jobs that are communication-intensive. This is due to the fact that memory-intensive jobs have more local data to work on in each processor and might not need to communicate very often. Therefore, it is unlikely for this combination of jobs to be seen in a real system.

The ASQT values for the processor allocation using modified RSR with user directives are almost always lower than the RSR scheme. In some cases when one of the two job characteristic variables mm and cm is equal to 0.9, the ASQT measured is slightly higher than the adaptive-scan scheme. This is because some of the jobs experience higher turnaround time due to the size-reduction. However the difference is always very small compared to the difference between adaptive-scan and the cases when other combination of jobs are simulated.

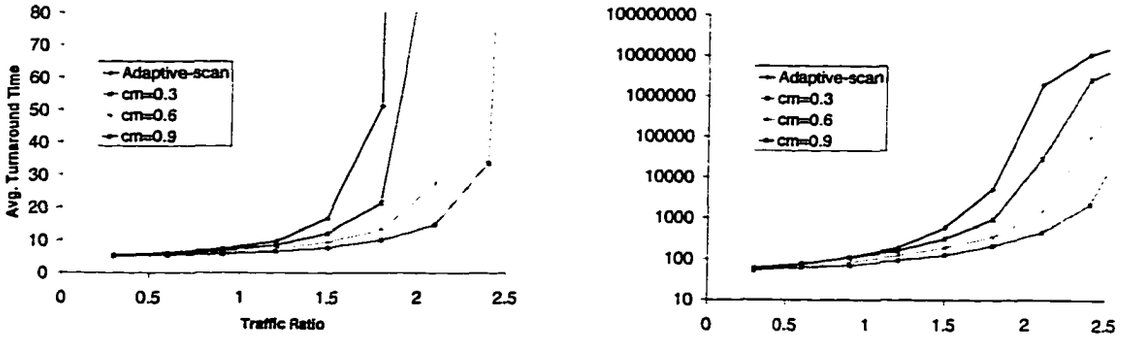
8.6 Discussion

This chapter studies the effect of user directives to processor allocation in multicomputer systems. Due to the high cost of performing size-reduction or non-contiguous allocation, user directives are useful in reducing the penalty caused by performing these



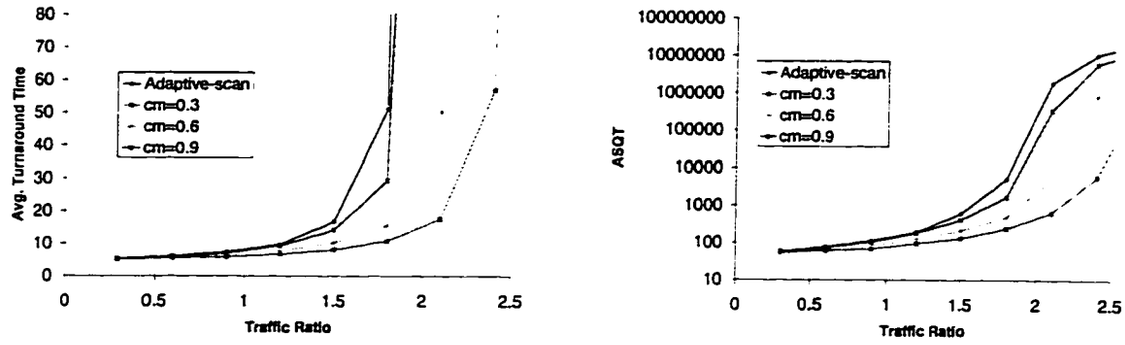
(a) Avg. Turnaround Time (b) ASQT

Figure 8.9 Using User Directives with Modified RSR (Normal Jobs, $mm = 0.3$).



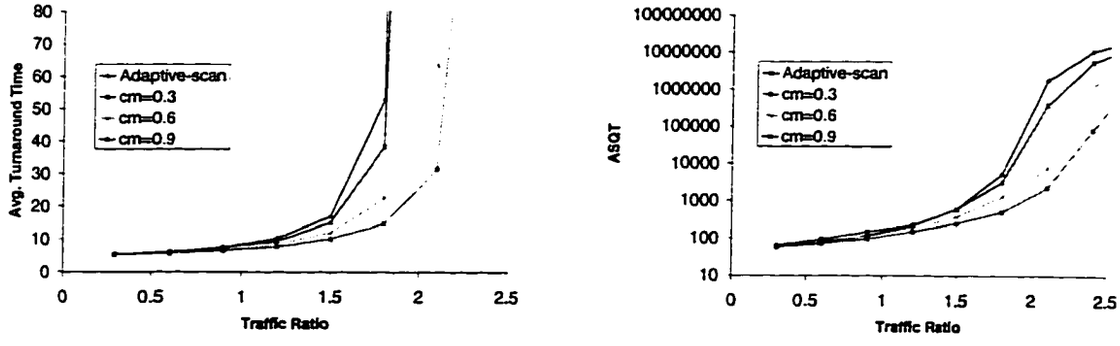
(a) Avg. Turnaround Time (b) ASQT

Figure 8.10 Using User Directives with Modified RSR (Normal Jobs, $mm = 0.6$).



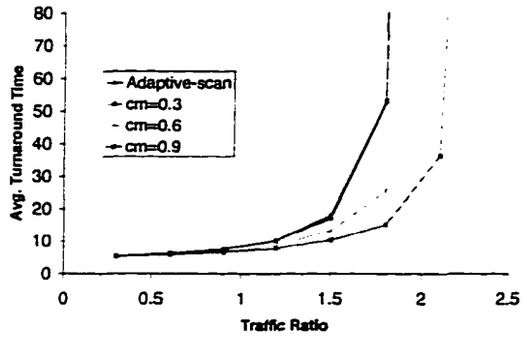
(a) Avg. Turnaround Time (b) ASQT

Figure 8.11 Using User Directives with Modified RSR (Normal Jobs, $mm = 0.9$).

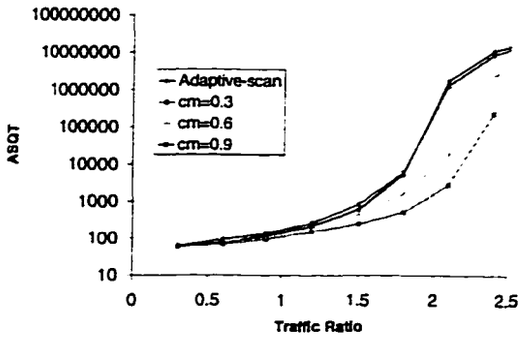


(a) Avg. Turnaround Time (b) ASQT

Figure 8.12 Using User Directives with Modified RSR (Uniform Jobs, $mm = 0.3$).

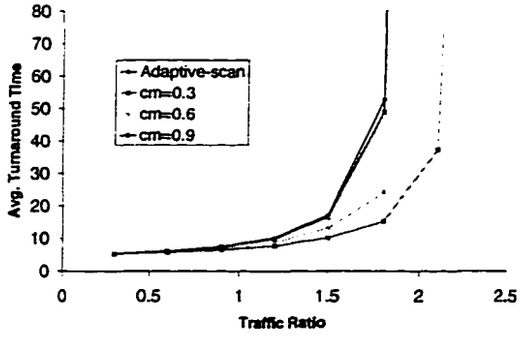


(a) Avg. Turnaround Time

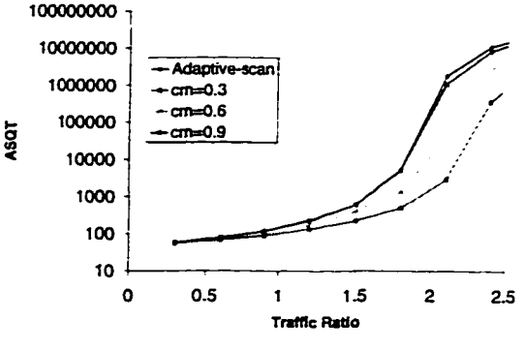


(b) ASQT vs. Traffic

Figure 8.13 Using User Directives with Modified RSR (Uniform Jobs, $mm = 0.6$).



(a) Avg. Turnaround Time



(b) ASQT

Figure 8.14 Using User Directives with Modified RSR (Uniform Jobs, $mm = 0.9$).

allocation alternatives on unfit jobs. Intuitively, the performance of the system is expected to improve if the user directives can be used to assist processor allocation. However, contradictory to expectation, using user directives for processor allocation does not always guarantee better system performance. In many cases, combining the conventional allocation along with RSR and ANCA even produces higher average turnaround time than using the conventional allocation algorithm alone.

The reason for the poor performance of the hybrid allocation to fail is a combination of the high cost associated with the RSR scheme and the lack of flexibility in choosing jobs for performing size-reductions. If a job is allocated with size-reduction, its execution time is increased extensively. If this job is placed in the middle of the system and a large job which is both memory and communication-intensive wants to enter the system, the incoming job will be blocked. Therefore, it is important to keep the size-reduced jobs away from the center of the mesh.

A modified RSR scheme is proposed to be used specifically in an environment where not all jobs can go through size-reduction. The modified RSR carefully allocates size-reduced jobs along the periphery of the mesh and therefore avoids the fragmentation. The hybrid allocation using modified RSR show significant performance improvement over the original hybrid allocation and constantly outperforms the conventional allocation scheme. It is therefore a good choice of processor management when the characteristics of jobs are specified.

9 CONCLUSIONS

9.1 Summary of the Proposed Processor Management Strategies

Multicomputers are cost-effective alternatives to the expensive conventional parallel machines. Because of the massive number of processing elements incorporated in building such systems, proper management of these computing resources is essential for the performance of the system.

In this dissertation, we have studied the performance issues of the processor management schemes for multicomputer systems. To improve the performance of such systems, several approaches can be taken. First, one can design better processors to use so that the computation time of tasks can be reduced. Alternatively, faster interconnection networks can be implemented so that the communication latency among processors is minimized. Both these approaches involve re-engineering and replacement of the hardware in the system and are costly solutions. However, the major problem associated with today's multicomputer systems is the underutilization of the computing resources caused by fragmentation of the processors. The above-mentioned hardware approaches to improve multicomputer performance do not solve the underutilization problem of the system. Instead of taking the expensive hardware approaches, this dissertation examines the performance problem in multicomputer systems from the software's point of view. Several processor management schemes are proposed to tackle the underutilization problem.

The restricted size-reduction scheme proposed in Chapter 3 allocates jobs to a smaller submesh when the requested submesh cannot be found. The flexibility of being able to allocate a job to a smaller submesh increases the possibility of allocating a job. In addition, when jobs can be allocated to smaller number of processors, the number of simultaneous jobs that the system can execute is increased. The penalty for this approach is the higher execution time for jobs that cannot obtain all the processors they requested.

Non-contiguous allocation has the advantage of being free of physical fragmentation. The problem associated with non-contiguous allocation is the potentially high communication latency incurred by jobs and the risk of saturating the intercommunication network. Advances in hardware technology such as wormhole switched interconnection networks have made the communication latency less sensitive to the distance between the source and destination nodes. Therefore, it is possible for non-contiguous allocation to take advantage of the faster interconnection network and become the processor allocation scheme of choice. We conducted an experimental study on the hypercube-based nCUBE2 multicomputer system to measure the communication latency when different processor allocation alternatives are applied. The results of this experiment reported in Chapter 4 indicate that, in addition to contiguity, the geometry of the allocated processors also has significant effect on affecting the communication latency of the job.

Based on the communication latency experiment, we propose the adaptive non-contiguous allocation scheme in Chapter 5. Contiguous allocation is always examined for the incoming job before non-contiguous allocation is performed. When non-contiguous allocation is necessary, our algorithm makes sure that partial contiguity and geometry is preserved in every allocated clusters of processors.

An attempt is also made to integrate the processor allocation and job scheduling in Chapter 6 to achieve the highest performance possible. An effective scheduling strategy based on the bypass-queue technique is proposed along with a simple processor alloca-

tion algorithm. Both the bypass-queue scheduling and the proposed fixed-orientation allocation have very low complexity and are very suitable to be implemented together.

A possible solution to the fragmentation problem is to migrate the allocated jobs toward one side of the system so the available processing units in the system are less fragmented and can be allocated to incoming tasks. Three different alternatives for performing such job migrations are proposed and studied in Chapter 7

The RSR and ANCA schemes are revolutionary approaches to processor allocation. The cost for performing these schemes is high if the jobs are memory-intensive or communication-intensive. It would be beneficial if user directives can be used to assist the system in deciding the most suitable processor allocation for a job. In Chapter 8, the effect of using user directives for a hybrid processor allocation is studied. The hybrid allocation combines the conventional allocation with the RSR and ANCA schemes and is expected to benefit from this combination. Contradictory to the intuitive expectation, the hybrid processor allocation does not guarantee a better performance. The reason for the poor performance is a combination of the high cost associated with the RSR scheme and the lack of flexibility in choosing the jobs to perform size-reduction or non-contiguous allocation. A modified RSR allocation is proposed to solve this problem in such an environment and the hybrid allocation using the modified RSR is evaluated.

Extensive simulation was conducted to evaluate the proposed processor management strategies. All the proposed schemes are capable of reducing the average turnaround time of a job and improving the operational range of the system. The RSR scheme is very efficient in increasing the operational range of the system with only a couple size-reductions. It always guarantees shorter average turnaround time for jobs when the system is under medium to high load. It has slightly worse performance when system load is low because of the penalty incurred for jobs that have their sizes reduced. The ANCA scheme outperforms the conventional processor allocation policies provided the communication latency of the jobs can be constrained. Even in the pessimistic scenario

when the cost of non-contiguous allocation is assumed to be unreasonably high, it still shows shorter average turnaround time if the adaptability of the system is high. In most cases, the ANCA scheme is able to provide better (or at least comparable) performance than using the conventional allocation algorithms alone.

Both the bypass-queue scheduling and the fixed-orientation allocation proposed in Chapter 6 have the ability to provide good overall system performance when applied alone. Combining the two strategies, even further performance improvement can be achieved. The low complexity to implement the bypass-queue and fixed-orientation scheduling makes the integration of the two schemes possible. The integrated scheme has the advantages of low complexity and performs all other conventional processor management strategies.

The job migration approach is limited by the size and shape constraints of the conventional allocation approach. Performing job migration and the allocation failures of incoming jobs is shown to provide worse performance than the conventional allocation. This is caused by the cost of migrating jobs in the system. Among the three alternatives for job migration, migration at job completion has the best performance because it can best utilize the available processors for the migration process.

Two hybrid allocation schemes are evaluated in an environment when user directives are given. The hybrid allocation which combines the conventional allocation with RSR and ANCA is shown to be performing worse than doing conventional allocation. The poor performance is caused by a combination of the high cost associated with the RSR and the lack of flexibility in performing size-reduction or non-contiguous allocation. Because of the lack of freedom in choosing jobs for size-reduction or non-contiguous allocation, a job which cannot be allocated using RSR or ANCA might be blocked by a size-reduced job in the system which is expected to stay in the system for a relatively longer period of time because of the size-reduction. If the size-reduced job is allocated in the middle of the mesh, the allocation of the incoming jobs becomes difficult. Another

hybrid allocation approaches uses a modified RSR allocation in conjunction with the conventional and ANCA allocation. The modified RSR only allocates size-reduced jobs along the periphery. The hybrid allocation using modified RSR is proven to outperform the conventional allocation under various combinations of memory-intensive and communication-intensive jobs in the system.

9.2 Concluding Remarks

The performance issue of multicomputer system is studied in this dissertation. The major challenge in improving the performance of such systems is to solve the low utilization of the computing resources caused by the inefficient processor management policies. Fragmentation and the consequent blocking effect are the main reasons for system underutilization.

Various processor management schemes are proposed and evaluated in this dissertation. The proposed schemes targeted the fragmentation problem from different perspectives. The RSR and ANCA schemes are revolutionary processor allocation approaches to utilize the available processors. The bypass-queue scheduling and the integrated processor management scheme solves the blocking problem. Job migration approach maintains the size and shape constraints of the jobs and attempts to improve the system performance by moving jobs in the system so the available processors are less fragmented.

All the proposed schemes are successful in improving the system performance by reducing the average turnaround time of jobs. No direct comparison among the proposed schemes is attempted in this dissertation because of the differences in the nature of these approaches. The difference in the nature of these processor management strategies makes the selection of a processor management policy important. Each scheme has its own limitation and may not be applicable under the condition that others can be used. For example, RSR is constrained by the memory requirement of the jobs and

ANCA is limited by the communication patterns. Choice of the proposed processor management strategies should be done on a cooperative rather than competitive manner. The selection of allocation methods should be based on the job's characteristics as studied in Chapter 8.

On top of the processor allocation, job scheduling is another way to improve the performance of multicomputer system. As illustrated in Chapter 6, the integration of processor allocation and job scheduling allows the system to benefit from both approaches. However, complexity associated with processor allocation and job scheduling is the main factor in designing an integrated policy. The proposed integrated policy has the advantage of low complexity. If the complexity of the allocation algorithm or the scheduling policy becomes less significant due to the advance of the processor design or the requirement of the jobs, integration of more complex and efficient allocation and scheduling strategies may produce further performance improvement.

9.3 Future Research

A number of studies that can be pursued in the future are listed here. First, most of the proposed processor management schemes are evaluated with the mesh architecture only. As other architectures such as the 3D mesh and MINs are gaining popularity, it is important to study the processor management strategies in those systems. A short-range goal will be developing algorithms for performing the proposed processor management strategies for other topologies.

Advancement in the hardware technology will affect the direction of research for the processor allocation algorithms. Non-contiguous allocation relies on a fast interconnection network and should be emphasised if future hardware design makes the communication latency immune to the distance between communicating nodes. When multithread processors and large local memory are used, the system will benefit from the concept of

size-reduction. In either case, more efficient algorithms are desired. Two important factors is involved in the development of future processor allocation algorithms, the system performance and the complexity to perform the algorithms. Algorithms that implement the concepts of size-reduction or non-contiguous allocation require a lot of refinement. Processor management schemes to use in an environment when jobs characteristics are known is another direction for future study.

Job scheduling in multiprocessor is a complicate task. Real-time scheduling introduces a greater challenge. How to combine processor allocation along with scheduling strategies to serve real-time jobs, or a combination of real-time and non-real-time jobs is an interesting topic for future research.

Essentially, the processor management schemes have to be incorporated into the operating system. The real implementation of requires understanding of the hardware, the processor management schemes, and the operating system. It is the ultimate goal of the study of the processor management strategies.

BIBLIOGRAPHY

[1] R. E. Kessler and J. L. Schwarzmeier, *"Cray T3D: A New Dimension for Cray Research,"* Proc. COMPCON, pp. 176-182, Feb. 1993.

[2] Cray, *Cray/MPP Announcement,* Cray Reserach Inc., Eagan, MN, 1992.

[3] Intel, *Paragon XP/S Product Overview,* Supercomputer System Division, Intel Corporation, Beaverton, OR, 1991.

[4] Intel, *A Touchstone DELTA System Description,* 1991.

[5] nCUBE, *nCUBE 6400 Processor Manual,* nCUBE Company, Beaverton, OR, 1990.

[6] C. B. Stunkel, D. G. Shea, D. G. Grice, P. H. Hochschild, and M. Tsao, *"The SP1 high-performance switch,"* Proc. 1994 Scalable High-Performance Computing Conference, pp. 150-157, May 1994.

[7] K. Li and K. H. Cheng, *"A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System,"* Journal of Parallel and Distributed Computing, 12, pp. 79-83, 1991.

[8] P. J. Chuang and N. F. Tzeng, *"Allocating Precise Submesh in Mesh-Connected Systems,"* IEEE Trans. on Parallel and Distributed Systems, pp. 211-217, Feb. 1994.

[9] Y. H. Zhu, *"Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers,"* Journal of Parallel and Distributed Computing, 16, pp. 328-337, 1992.

[10] J. Ding and L. N. Bhuyan, *"An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems,"* Proc. of Int. Conf on Parallel Processing, Vol. II, pp. 193-200, Aug. 1993.

[11] D. D. Sharma and D. K. Pradhan, *"A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers,"* Proc. of the 5th IEEE Symp. on Parallel and Distributed Processing, pp. 682-693, Dec. 1993.

[12] T. Liu, W-K. Huang, F. Lombardi, and L. N. Bhuyan, *"A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems,"* Proc. of Int. Conf. on Parallel Processing, vol. II. pp. 159-163, Aug. 1995.

[13] S. M. Yoo and H. Y. Youn, *"An Efficient Task Allocation Scheme for Two-Dimensional Mesh-Connected Systems,"* Proc. of the Int. Conf. on Distributed Computing Systems, pp. 501-508, May 1995.

[14] J. M. Chang, *"A High Performance Processor Allocation Strategy,"* Proc. of the Int. Conf. on Parallel and Distributed Computing Systems, pp. 110-114, Oct. 1997.

[15] B. S. Yoo and C. Das, *"Good Processor Management = Fast Allocation + Efficient Scheduling,"* Proc. of the 1997 Int. Conf. on Parallel Processing, pp. 280-287, Aug. 1997.

[16] P. D. Kulasinghe, *"A Distributed Submesh Allocation Scheme for Two-Dimensional Mesh-Connected Parallel Computers,"* Proc. of the Int. Conf. on Parallel and Distributed Computing Systems, pp. 115-119, Oct. 1997.

[17] K. C. Knowlton, *"A Fast Storage Allocator,"* Communications of ACM, vol. 8, pp. 623-625, Oct. 1965.

[18] M. S. Chen and K. G. Shin, *"Processor Allocation in an N-Cube Multiprocessor Using Gray Codes,"* IEEE Trans. on Computers, vol. C-36, NO. 12, pp. 1396-1407, Dec. 1987.

[19] J. Kim, C. R. Das, and W. Lin, *"A Top-Down Processor Allocation Scheme for Hypercube Computers,"* IEEE Trans. on Parallel and Distributed Systems, vol. 2, NO. 1, pp. 20-30, Jan. 1991.

[20] P. J. Chuang and N. F. Tzeng, *"Dynamic Processor Allocationin Hypercube Computers,"* Int. Symp. on Computer Architecture, pp. 40-49, May 1990.

[21] H. Wang and Q. Yang, *"Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors,"* Int. Conf. on Parallel Processing, pp. 25-32, Aug. 1991.

[22] W. Liu, V. M. Lo, K. Windisch, B. Nitzberg, *"Contiguous and Non-contiguous Processor Allocation Algorithms for k-ary n-cubes,"* Proceedingsof the 1995 International Conference on Parallel Processing, 1995.

[23] H. Choo, H. Y. Youn, G. Park, and B. Shirazi, *"Efficient Processor Allocation Scheme for Multi Dimensional Interconnection Networks,"* Proc. Int. Conf. on Parallel Processing, pp. 114-117, 1997.

[24] W. J. Dally, *"Performance Analysis of k-ary n-Cube Interconnection Networks,"* IEEE Trans. Computers, vol. 39 NO. 6, pp. 775-785, 1990.

[25] W. Liu, V. Lo, K. Windisch, and B. Nitzberg, *"Non-continuous Processor Allocation Algorithms for Distributed Memory Multicomputers,"* Proc. of 1994 Int. Conf. on Supercomputing, pp. 227-236, 1994.

[26] J. Mache, V. Lo and K. Windisch, *"Minimizing Message-Passing Contention in Fragmentation-Free Processor Allocation,"* Proc. of the Int. Conf. on Parallel and Distributed Computing Systems, pp. 120-124, Oct. 1997.

[27] D. D. Sharma, *"Processor Allocation in Hypercube Multicomputers: The Random Allocation Strategy,"* Proc. of the International Conf. on Parallel and Distributed Systems, pp. 439-445, Sep. 1995.

[28] D. Babbar and P. Krueger, *"A Performance Comparison of Processor Allocation and Job Scheduling Algorithms for Mesh-Connected Multicomputers,"* Proc. 6th Int. IEEE Symposium on Parallel and Distributed Processing, pp. 46-53, October, 1994.

[29] P. Krueger, T. Lai, and V. A. Dixit-Radiya, *"Job Scheduling is More Important than Processor Allocation for Hypercube Computers,"* IEEE Trans. on Parallel and Distributed Systems, 5, pp.488-197, May, 1994.

[30] P. Krueger, T. H. Lai, and V. A. Radiya, *"Processor Allocation vs. Job Scheduling on Hypercube Computers,"* Proc. Int. Conf. on Distributed Computing Systems, pp. 394-401, 1991.

[31] C. Yu, P. Mohapatra and C. R. Das, *"Processor Allocation Using a Reservation Techniques for Hypercube Computers,"* Proc. of Int. Conf. on Parallel and Distributed Computing and Systems, pp. 147-152, Oct. 1993.

[32] P. Mohapatra, C. Yu and C. R. Das, *"A Lazy Scheduling Scheme for Hypercube Computers,"* Journal of Parallel and Distributed Computing, 27, pp. 26-37, May, 1995.

[33] D. D. Sharma and D. K. Pradhan, *"Job Scheduling in Mesh Multicomputers,"* Proc. Int. Conf. on Parallel Processing, vol. II, pp. 251-258, 1994.

[34] D. Min and M. W. Mutka, *"Effects of Job Size Irregularity on the Dynamic Resource Scheduling of a 2-D Mesh Multicomputer,"* Proc. of PARLE '93, pp. 476-487, 1993.

[35] D. Min and M. W. Mutka, *"Efficient Job Scheduling in a Mesh Multicomputer Without Discrimination Against Large Jobs,"* Proc. of the IEEE Symposium on Parallel and Distributed Processing pp. 52-59, October, 1995.

[36] C. Y. Chang and P. Mohapatra, *"An Integrated Processor Management Policy for Mesh-Connected Multicomputer Systems,"* Int. Conf. on Parallel Processing, pp. 118-121, Aug. 1997.

[37] K. Suzaki, H. Tanuma, S. Hirano and Y. Ichisugi, *"A Time Sharing System Scheme That Uses a Partitioning Algorithm for Mesh-Connected Parallel Computers,"* Symposium on Parallel and Distributed Processing, 1995.

[38] B. S. Yoo, C. R. Das and C. Yu, *"Processor Management Techniques for Mesh-Connected Multiprocessors,"* Proc. of Int. Conf. on Parallel Processing, vol. II, pp. 105-112, Aug. 1995.

[39] D. E. Knuth, *The Art of Computer Programming, Volume 1, Fundamental Algorithms,* Addison-Wesley, 1973.

[40] C. Y. Chang and P. Mohapatra, *"An Adaptive Job Allocation Method for the Directly-Connected Multicomputer Systems,"* International Conference on Distributed Computing Systems, May, 1996.

[41] C. McCann and J. Zahorjan, *"Processor Allocation Policies for Message-Passing Parallel Computers,"* Proc. ACM SIGMETRICS Conf. pp. 19-32, May, 1994.

[42] C. Yu and C. R. Das, *"Limit Allocation: An Efficient Processor Management Scheme for Hypercubes,"* Int. Conf. on Parallel Processing, vol. II, pp. 143-150, 1994.

[43] Y. Zhu and M. Ahuja, *"On Job Scheduling on a Hypercube,"* IEEE Trans. on Parallel and Distributed Systems, pp. 62-69, January, 1993.

[44] K. Ramamritham, J. A. Stankovic, and P. Shiah, *"O(n) Scheduling Algorithms for Real-Time Multiprocessor Systems,"* Proc. Int. Conf. on Parallel Processing, pp. 143-152, 1989.

[45] K. Ramamritham, J. Stankovic, and P. Shiah, "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," IEEE Trans. on Parallel and Distributed Systems, pp. 184-194, April 1990.

[46] F. Wang, K. Ramamritham, and J. A. Stankovic, *"Bounds on the Performance of Heuristic Algorithms for Multiprocessor Scheduling of Hard Rea-Time Tasks,"* Proc. of the Real-Time Systems Symposium, pp. 136-135, 1992.

[47] D. Babbar and P. Krueger, *"On-Line Hard Real-time Scheduling of Parallel Tasks on Partitionable Multiprocessors,"* International Conference on Parallel Processing, vol. II, pp. 29 - 38, 1994.

[48] K. S. Hong and J. Y. T. Leung, *"On-Line Scheduling of Real-Time Tasks,"* IEEE Trans. on Computers, pp. 1326-1331, Oct. 1992.

[49] R. Krishnamurti and B. Narahari, *"Preemptive Scheduling of Independent Jobs on Partitionable Parallel Architectures,"* Int. Conf. on Parallel Processing, vol. I, pp. 268-275, 1992.

[50] N. G. Shivaratri and M. Singhal, *"A Transfer Policy for Global Scheduling Algorithms to Schedule Tasks With Deadlines,"* Proc. of 11th. Int. Conf. on Distributed Computing Systems, pp. 248 - 255, May 1991.

[51] J. A. Stankovic, M. Spuri, M. D. Natale, and G. C. Buttazzo, *"Implications of Classical Scheduling Results for Real-Time Systems,"* IEEE Computer, pp. 16-25, June 1995.

[52] O. Kwon, J. Kim, S. Hong, and S. Lee, *"Real-Time Job Scheduling in Hypercube Systems,"* Proc. Int. Conf. on Parallel Processing, pp. 166-169, August, 1997.

[53] M. S. Chen and K. G. Shin, *"Subcube Allocation and Task Migration in Hypercube Multiprocessors,"* IEEE Trans. on Computers, vol. 39, pp. 1146-1155, Sep. 1990.

[54] S. G. Akl, *The Design and Analysis of Parallel Algorithms,* Prentice Hall, 1989.

[55] C. Y. Chang and P. Mohapatra, *"Experimental Evaluation of Communication Latency in Multicomputer Systems,"* Int. Conference on Parallel and Distributed Computing Systems, pp. 163-166, Oct. 1997.

[56] J. H. Upadhyay, V. Varavithya and P. Mohapatra, *"Efficient and Balanced Adaptive Routing in Two-Dimensional Meshes,"* Proc. of the IEEE Symp. on High-Performance Computer Architecture, pp. 112-121, Jan. 1995.

[57] J. Upadhyay, V. Varavithya, and P. Mohapatra, *"Routing Algorithms for Torus Networks,"* Proc. Int. Conf. of High Performance Computing, New Delhi, India, pp. 743-748, December 1995.

[58] S. Q. Moore and L. M. Ni, *"The Effect of Network Contention on Processor Allocation Strategies,"* Proc. of the 1996 International Parallel Processing Symposium, pp. 268-274, April 1996.

[59] D. Min and M. W. Mutka, *"Effect of Job Interactions Due to Scattered Processor Allocations in 2-D Wormhole Networks,"* Proc. of Int. Conf. on Parallel and Distributed Computing Systems, pp. 262-267, Sep. 1995.

[60] I. D. Scherson and P. F. Corbett, *"Communication Overhead and the Expected Speedup of Multidimensional Mesh-Connected Parallel Processors,"* Journal of Parallel and Distributed Computing, pp. 86-96, Vol. 11, 1991.

[61] T. Eicken, D. E. Ceiller, S. C. Goldstein, and K. E. Schauser, *"Active Messages: A Mechanism for Integrated Communication and Computations,"* Int. Symposium on Computer Architecture, pp. 256-266, 1992.

[62] V. Karamcheti and A. A. Chien, *"A Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D,"* Int. Symposium on Computer Architecture, pp. 298-307, 1995.

[63] A. A. Chien and J. H. Kim, *"Planer-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors,"* Int. Symposium on Computer Architecture, pp. 268-277, 1992.

[64] C. J. Glass and L. M. Ni, *"The Turn Model for Adaptive Routing,"* Int. Symposium on Computer Architecture, pp. 279-287, 1992.

[65] *nCUBE2 Programmer's Guide,* nCUBE, 1992.

[66] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability,* McGraw-Hill, 1993.

[67] G. Amdahl, *"Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities,"* Proc. AFIPS Conf. pp. 483-485, 1967.

[68] X. H. Sun and L. M. Ni, *"Scalable Problems and Memory-Bound Speedup,"* Journal of Parallel and Distributed Computing, Vol. 19, pp. 27-37, 1993.

[69] S. T. Leutenegger and M. K. Vernon, *"The Performance of Multiprogrammed Multiprocessor Scheduling Policies,"* Proc. ACM SIGMETRICS Conf. pp. 226-236, 1990.

[70] J. L. Gustafson, *"Reevaluating Amdahl's Law,"* Comm. of ACM, pp. 532-533, May 1988.